BCM88690

Packet Processing

Programming Guide
BCM8869X
BCM8880X
BCM8848X
BCM8828X
BCM8883X

Table of Contents

Chapter 1: Overview

This document provides the packet processing driver reference for the following StrataDNX™ device families:

- BCM8869X (Jericho2)
- BCM8880X (J2C)
- BCM8848X (Q2A)
- BCM8828X (Q2U)
- BCM8883X (J2X)

Unless indicated otherwise, this document refers to the BCM88690, BCM88800, BCM88830, BCM88480, and BCM88280 devices, including derivatives and SKUs within each family. Note that all references to the BCM88670 (Jericho) refer to all legacy devices (such as the BCM88670 [Jericho], BCM88370 [QMX], BCM88470 [QAX], BCM88270 [QUX], and BCM88680 [Jericho+]).

NOTE:

- The BCM8828X device (Q2U) and BCM8829X (Q2N) are part of the BCM8848X family. In this document and in the SDK, all information related to the BCM88480 device also applies to the BCM88280 and BCM88290 devices unless explicitly stated otherwise.
- This document revision is aligned with the SDK 6.5.28 release.

1.1 Introduction

The packet processing driver reference is aligned to the SDK general concepts described in the *Traffic Manager Programming Guide* in the SDK General Concepts section.

This document maps the BCM88690 Packet-Processing (PP) architecture terms to the terms used within the BCM APIs. In addition, it provides the driver reference for all networking applications.

Chapter 2: Ports and Generalized Ports

Some BCM APIs include port parameters of type `bcm_port_t` and gport parameters of type `bcm_gport_t`. A `bcm_port_t` type typically refers to a port as a physical destination. Depending on the context, it can represent a local logical port, an aggregate port, or a system logical port. A gport is a data type that extends the notion of destination, covering entities such as logical ports.

Specifically, gports are used to capture:

- Global system port ID (singletons)
- System LAG/trunk ID
- Local port ID
- ModPort = pair of {Unit, Port}
- Logical interfaces
  - L2 attachment circuits – AC logical interfaces (LIFs)
  - L2 pseudowire emulation (PWE or MPLS_PORT) LIFs
  - Tunnels identifier – IP/MPLS
  - VXLAN LIFs (UDP over IP tunnel) tunnels
  - ARP
- Forward Equivalence Class (FEC), referred to as forward groups
- Virtual output queues (VOQs)
- Black hole (BCM_GPORT_BLACK_HOLE) – Packet drop

NOTE: It is recommended to use a *Trap Destination with Drop* action instead of a black hole gport. Black hole will be deprecated in next-generation devices.

NOTE: Black Hole will only work if the following configuration is applied:

Create a trunk, do not set it with any members. For example:

```
int unit = 0;
int groups_in_pool_1 = *dnxc_data_1d_get(unit, "trunk", "parameters", "pool_info",
"max_nof_groups_in_pool", 1);
int trunk_id;
BCM_TRUNK_ID_SET(trunk_id, 1, groups_in_pool_1 - 1);
print bcm_trunk_create(0, 0, &trunk_id);

/** Configure trunk info */
bcm_trunk_info_t trunk_info;
bcm_trunk_info_t_init(trunk_info);
trunk_info.psc = BCM_TRUNK_PSC_PORTFLOW;

/** Set trunk with PSC */
bcm_trunk_set(unit, trunk_id, &trunk_info, 0, NULL);
```

Set last flow aggregate to point to last VOQ. For example:

```
int nof_voqs = *dnxc_data_get(unit, "ipq", "queues", "nof_queues", NULL);
int nof_flow_aggs = *dnxc_data_get (unit, "trunk", "flow_agg", "nof_flow_aggs", NULL);
int flow_agg_id = nof_flow_aggs - 8;
bcm_flow_agg_info_t flow_agg_info;
BCM_GPORT_UNICAST_QUEUE_GROUP_SET(flow_agg_info.base_voq, nof_voqs - 8);
flow_agg_info.trunk_id = trunk_id;
flow_agg_info.nof_cosq_levels = 1;
```

```
bcm_trunk_flow_agg_set(unit, flow_agg_id, 0, &flow_agg_info);
```

- Trap destination
- CPU directly attached port (`BCM_GPORT_LOCAL_CPU`). Can be used as a reference to local CPU port with outgoing header type CPU. In case of several such ports configured on the device, the first one is used. In case there is no such port, the first port with type CPU is returned.

The `gport` is a structured type comprised of an object ID within the object name and allocation space, tagged with the object type, and sometimes including more information in a subtype encoding. To convert from the raw, integer-type object-ID to a (typed) `gport` and vice versa the BCM API provides a set of `gport` macros.

In BCM PP APIs, gport can be provided as a destination (such as when setting Flow-ID destination in MACT forwarding), as an L2/L3 gport object (such as VLAN-Port and MPLS-Port property APIs), or as a local-port entity processing within the PP core (for example, the `bcm_port_class_set` API).

NOTE: For many APIs, the port attribute may be a physical port, logical port, or both, depending on the context and documentation.

2.1 Packet Processing Ports

In BCM APIs, packet processing ports (PP ports) are directly derived from a local port or trunk or from outbound-mirror traffic. The PP port ID is managed internally by the SDK and is dynamically allocated (which means it is usually not a 1:1 mapping to Local-Port).

The following allocations exist for PP ports:

- On creation, Local-Port is assigned with one PP port.
- When a trunk is created, a number of PP ports are allocated (one per core). When a trunk member is added to trunk group, its PP port properties are lost and moved to be the same as the trunk port properties.
- Outbound-mirror has the same creation sequence as a regular local port and requires an additional incoming PP port.

PP ports do not have a notion in BCM APIs. Instead, to configure PP port properties, provide a gport of Local-Port or trunk. Internally, the SDK extracts PP port information from the given gport and sets it accordingly. Still, for cases of injection with PTCH, a PP port value must be known, and a `get` API is available to retrieve PP port values for the given Local-Port or trunk.

Similar to Local-Ports, PP ports support dynamic port provisioning, and its creation or deletion is decided according to regular port APIs.

When a new Local-Port or trunk is defined, a new PP port is created and initialized with some default settings. Creating a new Local-Port or trunk requires calling a set of APIs. For more information, refer to the Port Provisioning and Trunk (Link Aggregation) Ports chapters in the *Traffic Manager Programming Guide*.

The following steps are required to make a PP port functional, and they are supported per local port or trunk gport only (calling per trunk member is not possible):

1. Set the port header-type per direction. The following port header-types are available:
   - ETH – Supported in both directions. Set initial settings for the port to support Ethernet headers. For the incoming direction, the port will be added to E bitmap in port configuration (except when the port interface is not external, such as a mirror recycle port).
   - RAW – Supported in both directions. RAW processing allows static forwarding processing to be set, where the entire packet is considered as a payload. For the outgoing direction, the packet is transmitted unchanged. For more information, see Section 2.3, RAW Processing Ports.
   - RAW_WITH_ORIGINAL_SYSTEM_HEADERS – Supported in the outgoing direction only. The behavior is the same as the RAW header type, but the packet will go out with the original system headers.

- INJECTED_2_PP – Supported in the incoming direction only. For the incoming direction, the port supports at the start of the packet an injected header, PTCH2, only. With the PTCH2 header it is possible to set the incoming-PP port, which can be different from the incoming local port. In addition, PTCH2 has an indication of the appropriate next header type processing (ITMH vs. ETH) and Opaque-value that can be used later by the pipeline. According to PTCH2, parser will know how to parse the headers. The port will be added to E bitmap in port config.
- INJECTED_2_PP_JR1_MODE – Supported in the incoming direction only. The port is considered as an Ethernet port. Identical to INJECTED_2_PP expect in this port ITMH is in Jericho format. Supported only on devices that interoperate with legacy devices.
- INJECTED_2 – Supported in the incoming direction only. The port is the same as in INJECTED_2_PP. The only difference is that the port will not be added to E bitmap in port config, so it will not be considered as an Ethernet port.
- INJECTED_2_JR1_MODE – Supported in the incoming direction only. Identical to INJECTED_2 expect in this port ITMH is in Jericho format. Required for OAMP in Jericho interop mode. Supported only on devices that interoperate with legacy devices.
- INJECTED – Supported in both directions. In this case, the port supports an injected header (PTCH1 only) at the start of the packet. With the PTCH1 header, it is possible to set the source-system-port, which also sets the Incoming-PP-port and can be different from the incoming local port. In addition, PTCH1 provides properties similar to PTCH2 to indicate the appropriate next header type processing (ITMH or ETH) and Opaque-value. In the outgoing direction, the packet is encapsulated with the header as PTCH1 plus format. For the BCM88800 only, it is possible to change the PTCH-1 plus format to scale up the mapping between system-port to local-port at the egress first-pass (global configuration) and passing the pp-port field from the first-pass to the second-pass for recycled injected scenarios.
- RCH_0 – Supported in the incoming direction only. Set the initial settings for the port to parse the Recycle-header. The port will not be added to the E bitmap in the port configuration. In this mode, Source-System-Port will be copied from the first pass by the Recycle-header, and the Incoming PP-Port is derived from the RCY-port.
- RCH_1 – Supported in the incoming direction only. Set the initial settings for the port to parse the Recycle-header. The port will not be added to the E bitmap in the port configuration. Source-System-Port will be copied from the first pass by the Recycle-header. The incoming PP-Port is derived by a mapping from Source-System-Port. It is required to add System-Port to PP-Port mapping for the incoming port by using `bcm_port_control_set` with `bcmPortControlSystemPortInjectedMap`.
- RCH_SRV6_USP_PSP – Supported in both directions. Set the initial settings for the port to parse the recycled RCH_SRV6_USP_PSP header. The port will not be added to the “E” bitmap in the port configuration.
- CPU, ENCAP_EXTERNAL_CPU – Supported in the outgoing direction only. Required for directly attached (CPU) or remotely connected (Encap external CPU) CPU processing. In remotely connected CPU processing, it is expected that system-headers as they appear before being sent to the fabric, will be part of the outgoing packet. For more information refer to Section 2.2, CPU Processing Out-Ports.
- CPU_RATE_LIMITING – Supported in both directions. Required for additional processing (for example, rate limiting) for packets going to the CPU. It is typically used with RCY-Port. The outgoing direction behaves the same as the CPU but also prepends 2 bytes for information passed to the incoming direction. The incoming direction behaves similar to STACKING, but in this case, layer stack parsing starts from layer 3, which points to the forward layer from the first pass.
- TM – Supported in both directions. At the ingress, packets can be injected with an ITMH header only. (It is supported only in the JR2 ITMH format.) In this case, In-PP-port = local-port, and all packet processing resolution is resulted from the ITMH header. The port is not considered to be an Ethernet port. The same processing is done as in the `INJECTED_2` scenario except that the first header is ITMH and not PTCH2. At the egress, the packets are transmitted with an OTMH header only, OTMH base + OutLIF/CUD extension, or OTMH base + source port + OutLIF/CUD extension. The header format is configured by the SOC properties `tm_port_otmh_outlif_ext_mode` and `tm_port_otmh_src_ext_enable`.

NOTE: A global limitation is that TM in the outgoing direction cannot work with the TDM function.

- MPLS RAW – It is possible to configure a port to accept MPLS labels directly without an ETH header. For more information, see Section 2.4, MPLS Raw Processing Ports.

– STACKING – Supported in both directions. For the incoming direction, the entire packet is considered as payload. For the outgoing direction, the packet goes out with updated system-headers from the pipeline.

NOTE: The header-type is called *stacking,* but it is not for the stacking application. It is for other PP applications, such as egress-ingress multicast.

2. It is possible to skip parsing the first bytes of the packet. This allows injecting packets with user-defined headers without removing them before getting to the device. After ingress-processing, the device will remove this header before going to fabric.

NOTE: The following notes apply when skipping parsing:

- The functionality must be set only after the header-type is configured on certain port.
- The functionality is per TM port and not PP port. It means that any trunk port addition or removal must be updated with the required skip values. By default, skip is disabled (0 bytes).
- Only Ethernet, INJECTED, INJECTED_2, and INJECTED_2_PP header-types support this functionality.
- It is possible to set the first-header to be before or after the injected-header in the packet. By default, the first-header is before the injected-header. Only the INJECTED header-type can support both modes. Other header types support only the default mode.

3. If the port supports ETH packets, additional properties are expected to be set. For more information, see Section 21, Ethernet Bridge.
4. Static forwarding (force forward) is available per incoming-port. If used, all traffic should go to the same destination. For more information, see Section 2.3, RAW Processing Ports.
5. It is possible to configure outgoing port to drop all packets.

For injection traffic using the PTCH2 header, an Incoming-PP-port value is required. To do so, an API is available per local port or trunk to return its corresponding PP port.

When a local port is added or removed from the trunk or when a trunk group is destroyed, its PP ports are destroyed. It is expected that when this happens, it is the user's responsibility to free up all PP port properties prior to calling to the add, remove, or destroy operation. For example, if a VLAN port has its physical port located on a trunk, it is expected to call `bcm_vlan_port_destroy` before calling the API to destroy the trunk.

In addition, when a local port is removed from a certain trunk group, a new PP port is allocated and attached to it, and all its PP properties are reset to the same values as after SDK initialization. It is expected to follow the steps described in this section because a new local port is being created in the SDK.

NOTE: Removing a local port from a certain trunk group is not a traffic-safe action.

2.1.1 SOC Properties

- The following SOC properties configure the header-type of a port according to the type given on initialization:
  – `header_type_in_<port_num>=Type` (incoming direction)
  – `header_type_out_<port_num>=Type` (outgoing direction)

  The functionality is the same as setting the `bcmSwitchPortHeaderType` API. Type is the same as the suffix value of `BCM_SWITCH_PORT_HEADER_TYPE_*`.
- The following SOC property configures, per local port, the size of the first header to skip:

  `first_header_size_<port_num>=value`

  The functionality is the same as setting `bcm_port_control_set` with `bcmPortControlFirstHeaderSize` (and so its range values).

- The following SOC property configures the OUTLIF extension mode of the OTMH header:
  `tm_port_otmh_outlif_ext_mode=NEVER` (default), `IF_MC`, or `ALWAYS`
  - NEVER: Outlif extension is never added.
  - IF_MC: Outlif extension is added only for MC.
  - ALWAYS: Outlif extension is always added.
- The following SOC property configures the source port of the OTMH header:
  `tm_port_otmh_src_ext_enable=0` (default) or `1`
  - 0: Do not add the source port to the OTMH header.
  - 1: Add the source port to the OTMH header.

2.1.2 Configuration Flow

For the BCM88800 only, it is possible to change the PTCH-1 plus format to scale up the mapping of system-port to local-port for injected scenarios. This is done by calling the following:

`bcm_switch_control_set(unit, bcmSwitchInjectedHeaderMode, arg)`

- `arg=1` – The PTCH-1 plus format mode includes an additional field, which is the pp-port mapping.
- By default, `arg=0` (PTCH-1 plus without pp-port field).

To set Port's header-type, call:

`bcm_switch_control_indexed_port_set(unit, port, key, value)`

- `port` – Local-Port/Trunk
- `key.type` – bcmSwitchPortHeaderType
- `key.index` – 0 (both directions), 1 (incoming), 2 (outgoing)
- `value.value` (as indicated in the following table)

Port-Type	Value
ETH	BCM_SWITCH_PORT_HEADER_TYPE_ETH
RAW	BCM_SWITCH_PORT_HEADER_TYPE_RAW
RAW_WITH_ORIGINAL_SYSTEM_HEADERS	BCM_SWITCH_PORT_HEADER_TYPE_RAW_WITH_ORIGINAL_SYSTEM_HEADERS
INJECTED_2_PP	BCM_SWITCH_PORT_HEADER_TYPE_INJECTED_2_PP
INJECTED_2_PP_JR1_MODE	BCM_SWITCH_PORT_HEADER_TYPE_INJECTED_2_PP_JR1_MODE
INJECTED_2	BCM_SWITCH_PORT_HEADER_TYPE_INJECTED_2
INJECTED_2_JR1_MODE	BCM_SWITCH_PORT_HEADER_TYPE_INJECTED_2_JR1_MODE
INJECTED	BCM_SWITCH_PORT_HEADER_TYPE_INJECTED
RCH_0	BCM_SWITCH_PORT_HEADER_TYPE_RCH_0
RCH_1	BCM_SWITCH_PORT_HEADER_TYPE_RCH_1
RCH_SRV6_USP_PSP	BCM_SWITCH_PORT_HEADER_TYPE_RCH_SRV6_USP_PSP
CPU	BCM_SWITCH_PORT_HEADER_TYPE_CPU
Encap External CPU	BCM_SWITCH_PORT_HEADER_TYPE_ENCAP_EXTERNAL_CPU
CPU Rate Limiting	BCM_SWITCH_PORT_HEADER_TYPE_CPU_RATE_LIMITING
TM base	BCM_SWITCH_PORT_HEADER_TYPE_TM
MPLS RAW	BCM_SWITCH_PORT_HEADER_TYPE_MPLS_RAW
Stacking	BCM_SWITCH_PORT_HEADER_TYPE_STACKING

NOTE: The direction both is supported only on a switch header type value that supports both incoming and outgoing directions.

For Ethernet port properties, see Section 21.4, Port Bridging Properties.

To retrieve PP-port according to local-port, call:

`bcm_port_get(unit, port, flags, interface_info, mapping_info):`

- `port` – Local-Port

  Returns `mapping_info.pp_port`

To retrieve the PP port according to Trunk-gport call:

`bcm_trunk_pp_port_allocation_get(unit, trunk, flags, allocation_info):`

- `trunk` – Trunk-ID

Returns:

- `allocation_info.core_bitmap` – valid PP port per core
- `allocation_info.pp_port_per_core_array` – Per core-ID returns the PP port

To configure the size of the first header to skip before the injected header (per TM-port), call the following API:

`bcm_port_control_set(unit, port, type, value)`

- `port` – Local port or trunk member gport
- `type` – `bcmPortControlFirstHeaderSize`
- `value`

It is possible to skip the first header that comes immediately after the injected header. This is performed per TM-port. The size is defined by calling the following API:

`bcm_port_control_set(unit, port, type, value)`

- `port` – Local port or trunk member gport
- `type` – bcmPortControlFirstHeaderSizeAfterInjected
- `value`

This configuration is supported only for ports with port header type INJECTED.

To configure the dropping of all packets on outgoing port, call the following:

`bcm_port_control_set(unit, port, type, value)`

- `port` – Local port or trunk member gport
- `type` – `bcmPortControlDiscardEgress`
- `value` – When set to 0, it will be disabled; otherwise, it will drop all packets

To enable the mapping of a system port to a local port to be used for injected scenarios (PTCH1), call the following:

`bcm_port_control_set(unit, port, type, value)`

- `port` – Local port or trunk gport
- `type` – `bcmPortControlSystemPortInjectedMap` (ingress mapping) and/or `bcmPortControlEgressSystemPortInjectedMap` (egress mapping for the BCM88800. See the beginning of this section.)
- `value` – Use 1 to enable the mapping. Use 0 to clear the mapping

2.1.3 Shell Commands

None

2.1.4 Application Reference

None

2.2 CPU Processing Out-Ports

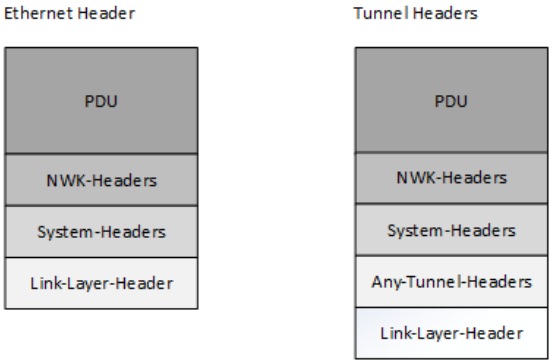
This section focuses on packet processing aspects related to outgoing ports with CPU header-types (CPU Processing Ports). CPU processing ports are ports expected to receive system-headers attached to the forward/trap/snoop packet. The SDK supports two kinds of CPU processing ports:

Header-Type	Interface
`CPU`	CPU is directly attached to CPU interface
`ENCAP_EXTERNAL_CPU`	CPU is externally attached to ETH interface

Packets transmitted to CPU processing out-ports expect to receive the original network-headers (that is, network-headers will not undergo egress packet processing) and system-headers as send to the fabric.

For the header type `ENCAP_EXTERNAL_CPU`, an additional Ethernet header or any tunnel header is constructed prior to the system-headers (that is, the packet is transmitted out as PayloadoSystem-HeadersoEth-Header or PayloadoSystem-HeadersoTunnel-Headers). The tunnel headers can be any tunnel encapsulation for getting through the network to the remote CPU. Traffic to the remote CPU port is encapsulated with an ETH (.1q) header or tunnel headers. A packet that is trapped, snooped, or forwarded to a CPU port on a valid LL OutLIF is encapsulated with an outer LL Ethernet header. A packet that is trapped, snooped, or forwarded to a CPU port on a valid tunnel OutLIF is encapsulated with the corresponding tunnel headers (for example, IPv4, IPv6, or MPLS). In this case, the packet received at the CPU looks similar to the packet shown in the following figure.

Figure 1: Outer LL Ethernet Header



The remote CPU network encapsulation can be set using standard routing into the overlay network encapsulation entries. For example, if it is an L2 switch network, the ETH-header encapsulation can be set using the standard L3 ARP entry with VLAN translation information (see Section 22.7, L3 Egress Object: Egress-ARP (Next-Hop)); while if it is an IPv4 network, the IPv4 tunnel encapsulations can be set using the standard IP tunnel entries (see Chapter 29, IP Tunnel v4 Encapsulation). The EtherType of the system headers for an ETH-header can be configured explicitly. All VLAN editing and QoS actions that are supported by L3 objects can be used for remote CPU routing as well.

2.2.1 SOC Properties

`header_type_out_<port_num>=Type (CPU/ENCAP_EXTERNAL_CPU)`. For more information, see Section 2.1, Packet Processing Ports.

2.2.2 Configuration Flow

1. Set the header type of the port (`bcm_switch_control_indexed_port_set(unit, port, key, value)`.
   - `port` – Local-Port/Trunk
   - `key.type` – `bcmSwitchPortHeaderType`
   - `key.index` – 2 (outgoing)
   - `value.value` – `BCM_SWITCH_PORT_HEADER_TYPE_CPU/ENCAP_EXTERNAL_CPU`

   For more information, see Section 2.1, Packet Processing Ports.
2. To set FTMH EtherType for remote CPU routing call `bcm_switch_control_set(unit, type,arg)`.
   - `type` – `bcmSwitchFtmhEtherType`
   - `arg` – FTMH EtherType
3. Configure the ARP object with VLAN editing. For more information, see Section 22.7, L3 Egress Object: Egress-ARP (Next-Hop).
4. Configure tunnel encapsulation objects. For more information, see Section 29.3, Egress L3 IP-Tunnel IPv4 Encapsulation Object, Section 31.3, Egress L3 IP-Tunnel IPv6 Encapsulation Object, and Chapter 25, MPLS LER Encapsulation.

2.2.3 Shell Commands

None

2.2.4 Application Reference

Example of remote CPU L2 encapsulation:

- Type: CINT reference
- Path: `$SDK/src/examples/sand/cint_l2_encap_external_cpu.c`

Example of remote CPU IP tunnel encapsulation:

- Type: CINT reference
- Path: `$SDK/src/examples/dnx/tunnel/cint_dnx_tunnel_l2_encap_external_cpu.c`
- Main Function: `dnx_tunnel_l2_external_cpu_run_all(...)`

2.3 RAW Processing Ports

This section focuses on packet processing aspects related to ports with RAW header-type. RAW processing ports are ports that are not expected to do any packet processing. The forwarding decisions are done according to trap information.

2.3.1 SOC Properties

`header_type_<port_num>=Type (RAW).`

2.3.2 Configuration Flow

1. Set the header-type of the port, `bcm_switch_control_indexed_port_set(unit, port, key, value)`.
   - `port` – Local-Port/Trunk
   - `key.type` – `bcmSwitchPortHeaderType`
   - `key.index` – 0 (both)
   - `value.value` – `BCM_SWITCH_PORT_HEADER_TYPE_RAW`

   For more information, see Section 2.1, Packet Processing Ports.
2. To set static forwarding per in local port, call: `bcm_port_force_forward_set(unit, port, egr_port, enable)`
   - `port` – In Local-Port
   - `egr_port` – Destination-Gport, can be one of the following:
     - Local-Port – Local port destination
     - System port gport – Create using `BCM_GPORT_SYSTEM_PORT_ID_SET`
     - Multicast group – Create using `BCM_GPORT_MCAST_SET`
     - VOQ gport – Create using `BCM_GPORT_UNICAST_QUEUE_GROUP_SET`
     - Ingress trap gport – Create using `BCM_GPORT_TRAP_SET`

NOTE: If the defined destination is not the ingress trap gport, the SDK allocates internally the user-defined trap, and updates its attributes to the required destination.

   - `enable` – 1 to enable static forwarding or 0 to disable

2.3.3 Shell Commands

None

2.4 MPLS Raw Processing Ports

In DNX devices, the base PP processing assumes that packets always have an Ethernet header as the outermost header when entering or leaving the device.

Working in MPLS raw mode on a selected port enables special processing, forcing operation on the underlying network layer protocol. The packet header starts from the MPLS label on traffic going in to or out of the port.

This mode is applicable for ports on which it is known in advance that they never act as bridge ports (that is, they do not process or filter based on the link layer). These ports should logically operate as if an Ethernet header exists to enable the network layer protocol.

NOTE: MPLS RAW is not supported in JR1 interoperability mode.

2.4.1 SOC Properties

None

2.4.2 Configuration Flow

Configure MPLS raw mode on selected ports by calling

`bcm_switch_control_port_set(unit, port, type, arg)`

- `port` – MPLS raw port

- `type` – `bcmSwitchPortHeaderType`
- `arg` – `BCM_SWITCH_PORT_HEADER_TYPE_MPLS_RAW`

If forwarding is ILM, configure an ILM forwarding entry by calling

`bcm_mpls_tunnel_switch_create(unit, info)`

- `info.egress_if` – 0
- `info.port` – Destination port
- `info.action` – `BCM_MPLS_SWITCH_ACTION_SWAP`

  In that case, the ILM forwarding entry does not have a valid ARP pointer. For other fields, see Section 23.6, MPLS Forwarding: ILM.

Other configurations are the same as those for general MPLS APIs usage.

NOTE:

- To avoid adding Ethernet encapsulation, MPLS RAW should not set an ARP ETPS entry or valid ARP pointer at the forwarding entry.
- If forwarding is L3 or L2, do not connect the MPLS tunnel entry to ARP.

2.4.3 Shell Commands

None

2.4.4 Application Reference

For an example of MPLS RAW:

- **Type**: CINT reference
- **Path**: `$SDK/src/examples/sand/cint_sand_mpls_lsr.c when setting cint_mpls_lsr_info.is_mpls_raw=1`

Chapter 3: L2 and L3 Ports (GPORTs)

3.1 GPORT Interfaces

In the BCM API, many objects have a software handle of gport type (`bcm_gport_t`). Depending on the context, it can represent a local logical object, an aggregation of objects, or a system logical object (see Chapter 2, Ports and Generalized Ports). A *gport* is a 32-bit data type that includes an object-ID within the object namespace and is tagged with an object type and properties to uniquely identify it.

This section describes gport entities that are used as an L2 or L3 logical interface:

- L2 Logical interfaces are referred to as L2-LIF and are interfaces of VSIs. L2-LIFs may be protected or forwarded using L2 FEC.
  - L2 attachment circuits – AC logical interfaces (VLAN_PORT)
  - L2 pseudowire emulation – PWE logical interfaces (MPLS_PORT)
  - VXLAN (UDP over IP tunnel) logical interface (VXLAN_PORT)
- L3 logical interfaces are referred to as L3-LIF:
  - IP-tunnels or IP-GRE tunnels (TUNNEL)
  - MPLS tunnels (TUNNEL)
  - ARP-pointer or Link-layer that represent a pointer to the next-hop MAC (TUNNEL)

For other types of GPORT interfaces, see Chapter 2, Ports and Generalized Ports.

3.2 Gport Encoding of a Logical Interface

The gport is 32 bits and usually includes four items:

1. Gport type (bits 31:26) – Specify the L2/L3 Gport interface to reference:
   - PWE: `BCM_GPORT_MPLS_PORT`
   - AC: `BCM_GPORT_VLAN_PORT`
   - VXLAN: `BCM_GPORT_VXLAN_PORT`
   - Tunnel: `BCM_GPORT_TYPE_TUNNEL`
   - Extender: `BCM_GPORT_EXTENDER_PORT`
2. Gport subtype (bits 25:22) – Specify the type of L2/L3 interface.
3. Gport additional information – Provide additional information to uniquely identify the gport handle.
4. Gport value – The L2/L3 Gport interface ID.

The following diagram summarizes the GPORT encoding for the different L2/L3 logical interfaces.

Figure 2: GPORT Encoding for L2/L3 Logical Interfaces

	31 26	25 22	21 20	19 18	17 16	4 3	0
	Gport Type (6)	Sub Type (4)	Additional info			Value	
1	VLAN	VIRTUAL INGRESS NATIVE	0			Virtual-ID (17)	
2	VLAN	VIRTUAL EGRESS DEFAULT	0			Virtual-ID (17)	
3	VLAN	VIRTUAL EGRESS MATCH	0			Virtual-ID (17)	
4	VLAN	VLAN_ TRANSLATION	0			Global-LIF-ID (18)	
5	MPLS/VLAN	LIF	EXC			Global-LIF-ID (20)	
6	VXLAN	Default	In-Global-LIF-ID (22)				
7	Tunnel	Deafult	Global-LIF-ID (22)				
8	Extender	LIF	Global-LIF-ID (22)				

The fields in Figure 2 are as follows:

- VLAN gport supports virtual type used to store ingress software handlers:
  - Gport subtype: `BCM_GPORT_SUB_TYPE_VIRTUAL_INGRESS_NATIVE` – The gport represents a native logical interface with no Global-LIF (also known as *glifless*).
  - Gport value: The value is the virtual software handle.
- VLAN gport supports virtual type used to store egress default ESEM software handlers:
  - Gport subtype: `BCM_GPORT_SUB_TYPE_VIRTUAL_EGRESS_DEFAULT` – The gport represents virtual handler to default ESEM result.
  - Gport value: The value is the virtual software handle.
- VLAN gport supports virtual type used to store egress ESEM software handlers:
  - Gport subtype: `BCM_GPORT_SUB_TYPE_VIRTUAL_EGRESS_MATCH` – The gport represents virtual handler to ESEM match entry.
  - Gport value: The value is the virtual software handle.
- VLAN gport supports Logical Interface (LIF) type that contains both ARP and AC information.
  - Gport subtype: `BCM_GPORT_SUB_TYPE_VLAN_TRANSLATION` – The gport represents ARP+AC LIF.
  - Gport value: The value is the Global-LIF-ID.
  - For more information about the creation of ARP+AC, see Section 22.7, L3 Egress Object: Egress-ARP (Next-Hop) with the `BCM_L3_FLAGS2_VLAN_TRANSLATION` flag. The typical configuration sequence that uses ARP+AC is usually LER encapsulation or VPLS.
- MPLS/VLAN gports supports L2 Logical Interface (LIF):
  - Gport subtype: `BCM_GPORT_SUB_TYPE_LIF` – The gport represents LIF.
  - Additional-information: For LIF, the object can be ingress/egress/ingress and egress LIF. Two bits of exclude information (EXC bits) are provided to indicate that:
    - `BCM_GPORT_SUB_TYPE_LIF_EXC_INGRESS_ONLY` – The LIF is configured only in ingress
    - `BCM_GPORT_SUB_TYPE_LIF_EXC_EGRESS_ONLY` – The LIF is configured only in egress
  - Gport value: The value is the Global-LIF-ID. Global LIF ID is limited to 20 bit value.
  - For more information about L2 logical interface, see Chapter 20, L2 Generalized Bridging Model, Chapter 27, Q-in-Q Bridging, and Chapter 26, VPLS and VPWS.

- VXLAN gport is handled by the incoming Global-LIF:
  - Gport subtype: Default.
  - Gport value: The value is the In-Global-LIF.
  - For more information about VXLAN port creation, see Chapter 32, VXLAN IP Overlay.
- Tunnel gport can be an IP tunnel or MPLS tunnel or ARP:
  - Gport subtype: Default
  - Gport value: The value is the Global-LIF-ID
  - For more information about the steps involved in tunnel creation, see Chapter 24, MPLS LER Termination, Chapter 28, IP Tunnel v4 Termination, and Chapter 30, IP Tunnel v6 Termination.
- The subtype *Egress Virtual Pointed* `BCM_GPORT_SUB_TYPE_EGRESS_VIRTUAL_POINTED`, is used for egress objects that do not allocate a global LIF and a GLEM mapping, and therefore, is expected to be pointed (linked) as part of EEDB LL only. This means that the local LIF is pointed directly from another local LIF. This subtype is supported for ARP and MPLS tunnels only.
  - It cannot be used by forwarding/multicast/ACL databases.
  - The only Gport type that can use it is Gport Tunnel.
- Extender gports support an L2 Logical Interface (LIF):
  - Gport subtype: `BCM_GPORT_SUB_TYPE_LIF` – The gport represents LIF.
  - Gport value: The value is the Global-LIF-ID.
- Push-profile:
  - Gport subtype `BCM_GPORT_SUB_TYPE_MPLS_PUSH_PROFILE` – the MPLS push command
  - Gport value – The value is the push profile ID
  - Used in BCM88670 interop mode. For more information, see Section 5.12, Push-Profile Allocation.

In addition, to simplify the GPORT encoding, SDK provides the following macros to handle the different GPORT subtypes:

Subtype indication:

- `BCM_GPORT_SUB_TYPE_IS_LIF(_gport)`
- `BCM_GPORT_SUB_TYPE_IS_VLAN_TRANSLATION(_gport)`
- `BCM_GPORT_SUB_TYPE_IS_VIRTUAL_INGRESS_NATIVE(_gport)`
- `BCM_GPORT_SUB_TYPE_IS_VIRTUAL_EGRESS_DEFAULT(_gport)`
- `BCM_GPORT_SUB_TYPE_IS_VIRTUAL_EGRESS_MATCH(_gport)`

Subtype information:

- `BCM_GPORT_SUB_TYPE_GET(_gport)`

Subtype LIF macros:

- `BCM_GPORT_SUB_TYPE_LIF_VAL_GET(_gport)` – Get LIF ID of gport of subtype LIF
- `BCM_GPORT_SUB_TYPE_LIF_EXC_GET(_gport)` – Get exclude information of gport of subtype Lif
- `BCM_GPORT_SUB_TYPE_LIF_SET(_gport, _exc_val, _lif_val)` – Create gport of subtype Lif using given exclude bit value and LIF ID.

Subtype Virtual macros:

- `BCM_GPORT_SUB_TYPE_VIRTUAL_INGRESS_NATIVE_GET(_gport)` – Get ID of gport of subtype virtual ingress native
- `BCM_GPORT_SUB_TYPE_VIRTUAL_INGRESS_NATIVE_SET(_gport, _virtual_id)` – Create gport of subtype virtual ingress native using given virtual ID.
- `BCM_GPORT_SUB_TYPE_VIRTUAL_EGRESS_DEFAULT_GET(_gport)` – Get ID of gport of subtype virtual egress default

- `BCM_GPORT_SUB_TYPE_VIRTUAL_EGRESS_DEFAULT_SET(_gport, _virtual_id)` – Create gport of subtype virtual egress default using given virtual ID.
- `BCM_GPORT_SUB_TYPE_VIRTUAL_EGRESS_MATCH_GET(_gport)` – Get ID of gport of subtype virtual egress match
- `BCM_GPORT_SUB_TYPE_VIRTUAL_EGRESS_MATCH_SET(_gport, _virtual_id)` – Create gport of subtype virtual egress match using given virtual ID.

Subtype VLAN translation macros:

- `BCM_GPORT_SUB_TYPE_L3_VLAN_TRANSLATION_GET(_gport)` – Get ID of gport of subtype VLAN translation
- `BCM_GPORT_SUB_TYPE_L3_VLAN_TRANSLATION_SET(_gport, _virtual_id)` – Create gport of subtype VLAN translation using given ID.

Subtype Egress Virtual Pointed macros:

- `BCM_GPORT_SUB_TYPE_VIRTUAL_EGRESS_POINTED_GET(_gport)` – Get ID of gport of subtype virtual egress pointed
- `BCM_GPORT_SUB_TYPE_VIRTUAL_EGRESS_POINTED_SET(_gport, _virtual_id)` – Create gport of subtype virtual egress pointed using given ID.

3.3 L3 Interface

L3 objects of type `bcm_if_t` can be divided into three types:

- `BCM_L3_ITF_TYPE_RIF` (Router interface), created by `bcm_l3_intf_create`
- `BCM_L3_ITF_TYPE_FEC`, created by `bcm_l3_egress_create` , object ID is the `if_id`
- `BCM_L3_ITF_TYPE_LIF`, may be created by:
  - Tunnels APIs (MPLS, IP)
  - ARP pointer created by `bcm_l3_egress_create`, object ID is the `encap_id`.
  - Virtual Egress Pointed – See Section 3.3.1, L3 Interface – Subtypes.

The following macros can be used to handle L3 objects:

- `BCM_L3_ITF_TYPE_IS_FEC(_l3_itf)`
- `BCM_L3_ITF_TYPE_IS_RIF(_l3_itf)`
- `BCM_L3_ITF_TYPE_IS_LIF(_l3_itf)`
- `BCM_L3_ITF_VAL_GET(_l3_itf)` – Get the value of L3 object
- `BCM_L3_ITF_SET(_l3_itf, _type, _id)` – Set L3 interface according to given type and value

In some cases, L3 objects (`bcm_if_t`) have to be translated to and from gport encoding (`bcm_gport_t`) to serve an input to other APIs.

The following macros allow this translation:

- `BCM_L3_ITF_FEC_TO_GPORT_FORWARD_GROUP(_gport_forward_group, _l3_itf_fec)` – Translate L3 object from L3 interface of type FEC to gport encoding. Gport is of type FORWARD.

  For example, this macro should be used for a forward group that is used as a destination in some APIs, defined using the port parameter of type `bcm_gport_t`. In this case the L3 interface received by `bcm_l3_egress_create` must first be converted to `bcm_gport_t` of type `BCM_GPORT_FORWARD_PORT`.
- `BCM_GPORT_FORWARD_GROUP_TO_L3_ITF_FEC(_l3_itf_fec, _gport_forward_group)` – Translate L3 object from gport encoding to L3 interface encoding. Gport is of type FORWARD, L3 interface is of type FEC.
- `BCM_L3_ITF_LIF_TO_GPORT_TUNNEL(_gport_tunnel, _l3_itf_lif)` – Translate L3 object from L3 interface of type LIF to gport encoding. Gport is of type TUNNEL.
- `BCM_GPORT_TUNNEL_TO_L3_ITF_LIF(_l3_itf_lif, _gport_tunnel)` – Translate L3 object from gport encoding to L3 interface encoding. Gport is of type TUNNEL, L3 interface is of type LIF.

3.3.1 L3 Interface – Subtypes

The L3 objects of type `bcm_if_t` have subtype definitions. The general APIs for subtypes are:

- `BCM_L3_ITF_SUB_TYPE_IS_SET(_l3_itf)` – Returns TRUE (1) if a subtype is set on the interface. Otherwise, it returns FALSE (0).
- `BCM_L3_ITF_SUB_TYPE_GET(_l3_itf)` – Returns the subtype value.
- `BCM_L3_ITF_SUB_TYPE_VALUE_GET(_l3_itf)` – Returns the element object value, without subtype or type encoding.

Below are the MACROs per each subtype, which are used to identify subtype, set the subtype encoding, and get the object value.

NOTE: When encoding a `bcm_l3_if` object with a subtype, the sequence is:

1. Call the subtype encoding MACRO (that is, `BCM_L3_ITF_SUB_TYPE_VIRTUAL_EGRESS_POINTED_SET`).
2. Call `BCM_L3_ITF_SET with BCM_L3_ITF_TYPE_LIF`.

Supported subtypes:

- `BCM_L3_ITF_TYPE_LIF` subtypes
  - Virtual Egress Pointed (`BCM_L3_ITF_SUB_TYPE_VIRTUAL_EGRESS_POINTED`):

    ```
    BCM_L3_ITF_SUB_TYPE_IS_VIRTUAL_EGRESS_POINTED(_l3_itf)
    BCM_L3_ITF_SUB_TYPE_VIRTUAL_EGRESS_POINTED_GET(_l3_itf)
    BCM_L3_ITF_SUB_TYPE_VIRTUAL_EGRESS_POINTED_SET(_l3_itf, _virtual_egress_pointed_val)
    ```

NOTE: The macro that translates L3 interfaces to gport (and vice versa), keeps the `EGR_POINTED` encoding for when translating an L3 interface of type LIF into gport of type TUNNEL (and vice versa).

3.4 ENCAP FORWARD ID Interface

NOTE: Using the ENCAP FORWARD ID interface is required only for Interop with BCM88670. For more information on this mode, see Chapter 5, Interoperability with Legacy Devices.

The encap_forward_id is used in L2 forwarding MACT `(bcm_l2_addr_*)` and cross-connect APIs `(bcm_vswitch_cross_connect_*)` to signal forwarding information in addition to port information. This is useful if a multidevice system is required and objects may be created by one device and only pointed by other devices.

The `encap_forward_id` is a forward handle pointer and used in addition to physical handle pointer to indicate the forwarding decision even if the local device (and local SDK) do not include any software information about those pointers.

Following the encoding:

- Type:

  `BCM_FORWARD_ENCAP_ID_TYPE_OUTLIF` – Forwarding information is Global-OutLIF

  `BCM_FORWARD_ENCAP_ID_TYPE_EEI` – Forwarding information is Egress Encapsulation Information (EEI):

  The following macros are used for encap_id type detection:

  `BCM_FORWARD_ENCAP_ID_IS_OUTLIF(_encap_id)`

  `BCM_FORWARD_ENCAP_ID_IS_EEI(_encap_id)`
- Value: Each type can be encoded with different attributes:

  OutLIF usage:

  `BCM_FORWARD_ENCAP_ID_OUTLIF_USAGE_GENERAL`

  EEI usage:

  `BCM_FORWARD_ENCAP_ID_EEI_USAGE_MPLS_PORT` – MPLS VC label and push profile

  `BCM_FORWARD_ENCAP_ID_EEI_USAGE_ENCAP_POINTER` – pointer to Egress Encapsulation database

  In this case, the following macros are used to handle the `encap_id` value:

  `BCM_FORWARD_ENCAP_ID_ EEI_USAGE_MPLS_PORT_SET(_forward_encap_id, _label, _push_profile)`

  `BCM_FORWARD_ENCAP_ID_ EEI_USAGE_MPLS_PORT_VC_GET(_forward_encap_id)`

  `BCM_FORWARD_ENCAP_ID_ EEI_USAGE_MPLS_PORT_PUSH_PROFILE_GET(_forward_encap_id)`

  In additional, general macros for `forward_encap_id` handling are provided:

  `BCM_FORWARD_ENCAP_ID_OUTLIF_USAGE_GET(_forward_encap_id)` – Get the usage of the `encap_id` of type OutLIF

  `BCM_FORWARD_ENCAP_ID_EEI_USAGE_GET(_forward_encap_id)` – Get the usage of the `encap_id` of type EEI

  `BCM_FORWARD_ENCAP_ID_VAL_GET(_forward_encap_id)` – Get the value of encap ID

  `BCM_FORWARD_ENCAP_ID_VAL_SET(_forward_encap_id, _type, _usage, _val)` – Set encap ID according to given type, usage, and value.

Chapter 4: MC Encapsulation Extension

4.1 Introduction

The multicast (MC) encapsulation extension database is located at the egress device and is used if more than a single encapsulation object (for example, OutLIF) is required, which occurs with various network applications. One common use case is Routing over Overlay (ROO) multicast flows, where more than one encapsulation object is required for multicast. The first pointer is ETH-RIF, and the second pointer (and maybe the third) is used for tunnels (for example, VXLAN tunnel and ARP or PWE tunnel). Another common application is in VPLS multicast where the first pointer is the PWE tunnel, and the second pointer is the MPLS tunnel.

Basic multicast capabilities are possible without using the MC encapsulation extension database. If multicasting requires only one encapsulation object (for example, OutLIF) or less, the MC encapsulation extension database is not necessary.

To use the MC replication extension database, an encapsulation extension object must be created. The object-ID is in the namespace of the Global-LIF field in system-headers and resides within its range. Thus, the object-ID reserved range is 0x300000-0x3BFFFF. The object-ID is used as a key to the encapsulation extension table, and the matching entry in the matching table provides two or three Global-LIF IDs.

To point to the MC replication extension object, an entry in the multicast database is required with a valid pointer as a parameter `encap_id`.

NOTE: The egress ACL feature also uses the MC replication extension database, and it is possible to point to the database from that feature. For more information, see Chapter 11, Field Processor.

4.2 Configuration Flow

To create an encapsulation extension, call

```
bcm_multicast_encap_extension_create(unit, flags, multicast_replication_index,
encap_extension_count,encap_extension_array);
```

– `flags` – Supported flags are:

Flag	Description
`BCM_MULTICAST_ENCAP_EXTENSION_WITH_ID`	The MC replication index will be the value specified in the encap_id. If this flag is omitted the MC replication index will be allocated and returned using the `encap_id`.
`BCM_MULTICAST_ENCAP_EXTENSION_REPLACE`	Replace the entry that is placed under the `encap_id` (must be used with the `BCM_MULTICAST_ENCAP_EXTENSION_WITH_ID` flag).

– `multicast_replication_index` – The MC replication index (ranged 0x300000-0x3BFFFF).
If the `BCM_MULTICAST_ENCAP_EXTENSION_WITH_ID` flag is set, the `multicast_replication_index` field needs to hold a valid value within the range. If the flag is omitted, the MC replication index is allocated and returned using this field.

– `encap_extension_count` – The size of the `encap_extension_array`.

– `encap_extension_array` – An array that holds encapsulation objects to be placed in the PPMC table. They should have INTF-LIF interface encoding while the first one can have INTF-RIF encoding.

4.3 Shell Commands

None

4.4 Application Reference

Example of replication and encapsulation extension usage:

- **Type:** CINT reference
- **Path:** `$SDK/src/examples/dnx/multicast/cint_multicast_pp_basic.c`

4.5 API Descriptions

API Name	Highlights
`bcm_multicast_encap_extension_create()`	Creates or updates an entry in the PPMC table for encapsulation extension.
`bcm_multicast_encap_extension_destroy()`	Removes an encapsulation extension entry from the PPMC table.
`bcm_multicast_encap_extension_get()`	Retrieve an encapsulation extension entry from the PPMC table.
`bcm_multicast_encap_extension_delete_all()`	Removes all the encapsulation extension entries from the PPMC table.
`bcm_multicast_encap_extension_traverse()`	Traverse the encapsulation extension entries from the PPMC table.

Chapter 5: Interoperability with Legacy Devices

5.1 Introduction

Connectivity with legacy devices is possible and allows existing systems to be upgraded in their line cards with new devices. The BCM88690 can be connected with legacy devices from BCM88670 and newer (BCM88670/BCM88476/BCM88680). Once the BCM88690 is connected with legacy devices, its functionality and scale is *downgraded* according to the functionality of the legacy device it is connected to. This means that new features in the BCM88690 cannot be enabled once it is connected to legacy devices. This is also true for object scaling of the device. Once downgraded, the BCM88690 is required to work with BCM88670 system headers. Special hardware is used to map between BCM88690 objects and BCM88670 system headers, and vice versa.

A device mode determines whether the BCM88690 is working with legacy devices. According to the device mode, special logic is applied that sets the device to support BCM88670 system headers. This document refers to working with BCM88670 connectivity as *IOP mode* or *BCM88670 interop mode*.

NOTE: If the device mode is set to work with legacy devices (BCM88670 interop mode), a different programmability loading image is required.

5.2 References

For additional information about interoperability with other devices, refer to the System Headers section in the *Packet Processing Architecture* document (88690-DG2xx).

5.3 System Headers Global Configuration

According to global configuration, the device can support BCM88690 system headers or legacy device (BCM88670) system headers.

For MAC learning and OAMP, different SOC properties are required to define port interfaces. See the configuration example in Section 5.3.1, SOC Properties.

In addition the BCM88670 system headers introduce a few options to the include/exclude internal headers:

- FTMH Load-Balancing-Key extension – Once set, it is possible to pass the load-balancing key in FTMH. This adds an additional byte to the FTMH header. This is useful for LAG MC pruning and network headers encapsulation marking. For example, UDP Src-Port for VXLAN header and Entropy label in MPLS header.
- FTMH DSP extension – This extension enables sending data to all the packets from the ingress to the egress using a two-byte `FTMH.Destination-Port-Extension`.
- FTMH Stacking extension – This extension enables sending data from the ingress to the egress with ACL using a two-byte stacking extension of the FTMH header.

NOTE: Other legacy usages such as stacking applications are not supported.

- User-Define header (UDH) – UDH extension header is a buffer filled by the user and sent from the ingress to the egress. This enables sending required data from the ingress to the egress by setting actions in the iPMF stages which fill in the UDH buffer. On start-up the size of UDH extension for all packets (0-8 bytes) is decided. At the ingress there are two such buffers. Their size is between 0B to 4B (1B granularity), and they are filled in by `bcmFieldActionUDHData0` and `bcmFieldActionUDHData1` actions in the iPMF stages.

  In the egress, reading the data depends on the size selected. 4 MSB (starting UDH1 buffer) uses `bcmFieldQualifyUDHData3` and the rest of the data uses `bcmFieldQualifyUDHData2`. If the size of both buffers in the ingress is less than 4B, only `bcmFieldQualifyUDHData3` is used to read the data.
- Learn information extension –Once set, 4 bytes of learn information is passed from the ingress pipeline to the egress pipeline. It is used for user-data (for example, the EVPN application).

5.3.1 SOC Properties

- The SOC property `system_headers_mode` enables BCM88690 system headers (1) or legacy devices (0). By default, the device works with BCM88690 system headers.
- The SOC property `programmability_ucode_relative_path`.

  For example, the `pemla/ucode/standard_1/jr2-comp-jr1-mode/u_code_db2pem.txt` path of the ucode file.
- The SOC property `system_ftmh_load_balancing_ext_mode` is used for Load-Balancing key existance. Options: Enabled/Disabled. By default, the device does not work without this extension.
- The SOC property `stacking_extension_enable` is used to set FTMH stacking extension Enable/Disable. By default, the device works without this extension.
- The SOC property `ftmh_dsp_extension_add` is used for FTMH DSP extensions. Options: Enabled/Disabled. By default, the device works without this extension.
- The SOC properties `field_class_id_size_0-3` are used to set the UDH0-1 buffers size in the following way (the size is in bits and the options for both buffer sizes are: 0/8/16/24/32):
  - UDH0 buffer size = `field_class_id_size_0` + `field_class_id_size_2`
  - UDH1 buffer size = `field_class_id_size_1` + `field_class_id_size_3`
- The SOC property `bcm886xx_pph_learn_extension_disable` is used for allowing the learn information extension. Options: True or false. By default, the device works without this extension.
- A workaround for the Recycle-Port for learning is required, for example: `ucode_port_41.BCM8869X=RCY.0:core_0.41`. For more information, see Section 5.13, L2 Learning.

- The OLP interface requires more than one port for learning. The BCM8869X_A0 device requires two OLP ports (one port in non-interop mode). The BCM8869X_B0 device requires four OLP ports (two in non-interop mode). For more information on the OLP interface, see Section 5.13, L2 Learning.

  A configuration example of BCM88690_B0 is as follows:

```
ucode_port_240.BCM8869X=OLP.0:core_0.240
ucode_port_241.BCM8869X=OLP.1:core_0.241
ucode_port_242.BCM8869X_B0=OLP.2:core_0.242
ucode_port_243.BCM8869X_B0=OLP.3:core_0.243
tm_port_header_type_in_240.BCM8869X=INJECTED_2
tm_port_header_type_in_241.BCM8869X=INJECTED_2
tm_port_header_type_in_242.BCM8869X_B0=INJECTED_2
tm_port_header_type_in_243.BCM8869X_B0=INJECTED_2
```

- The OAMP header-type is different in interop mode. For more information, see Section 5.10, Traffic Injecting. A configuration example is as follows:

```
config delete tm_port_header_type_in_232.*
config delete tm_port_header_type_in_233.*
config add tm_port_header_type_in_232.BCM8869X=INJECTED_2_PP_JR1_MODE
config add tm_port_header_type_in_233.BCM8869X=INJECTED_2_PP_JR1_MODE
```

5.3.2 Configuration Flow

None

5.3.3 Shell Commands

None

5.3.4 Application Reference

For CINT reference examples, folders are organized as follows:

- The following folder supports BCM88670 (JR1) devices only: `$SDK/src/examples/dpp`
- The following folder supports the BCM88670 family (DPP devices) and BCM88690 family (DNX devices): `$SDK/src/examples/sand`

NOTE: The BCM88690 family is usually supported in both system header modes. In addition, it is provided as a good example of how to port application code from the BCM88670 family to the BCM88690 family. For information highlighting the main difference between the BCM88670 family SDK and BCM88690 family SDK, refer to *Packet Processing Software Backward Compatibility with the BCM88670 and BCM88680* (88670-88690-AN2xx).

- The following folder supports the BCM88690 family only (DNX devices): `$SDK/src/examples/dnx`.

5.4 System Object Scale

Connectivity with legacy devices reduces the scale of system objects:

- Global-LIF ID range is limited to 256K.
- VSI range is limited to 32K.
- In-RIF range is limited to 32K.
- Out-RIF range is 4K by default and can be set up to 32K.
- Trunk Pool ID is limited to 0.
- Ingress VLAN edit action IDs range from 0 to 63.

5.4.1 SOC Properties

SOC property `rif_id_max` can be used to change the Out-RIF limitation.

5.4.2 Configuration Flow

None

5.4.3 Shell Commands

None

5.4.4 Application Reference

None

5.5 Port Properties

When working in interop-mode, the following changes exist:

- Unknown-DA MACs filtering/action can be defined per outgoing-port.

5.5.1 SOC Properties

None

5.5.2 Configuration Flow

To define per port, Unknown-DA egress filtering:

- Filter UC packets with unknown DA.
- Filter MC packets with unknown DA.
- Filter BC packets with unknown DA.

Call `bcm_port_control_set` with type `bcmPortControlEgressFilterDisable`. The arg is a bitmap of disable filters: `BCM_PORT_CONTROL_FILTER_DISABLE_UNKNOWN_DA_UC`, `BCM_PORT_CONTROL_FILTER_DISABLE_UNKNOWN_DA_MC`, `BCM_PORT_CONTROL_FILTER_DISABLE_DA_BC`.

If an egress filter is applied, an egress trap `bcmRxTrapEgUnknownDa` is invoked.

By default, port unknown DA filters are disabled.

5.5.3 Shell Commands

None

5.5.4 Application Reference

Usage examples for Unknown-DA per port:

- Type – CINT reference
- Path – `$SDK\src\examples\dpp\cint_port_egress_filter_example.c`

5.6 Host Table EEI Construction

The L3 host table could support EEI in its result. In such cases, EEI could be used for two purposes:

- As an additional EEDB pointer (IOP mode only)
- As an MPLS push or swap command

In both cases, EEI is resolved from the result. It is expected that L3 egress (FEC) does not use EEI as a result so that an override will not occur.

5.6.1 SOC Properties

None

5.6.2 Configuration Flow

Add Host entry by calling the API `bcm_l3_host_add(unit, info)`.

1. `info.l3a_flags: BCM_L3_ENCAP_SPACE_OPTIMIZED`
2. `info.encap_id` – Either EEI as MPLS push command (0 in non-interoperability mode) when its type is `BCM_FORWARD_ENCAP_ID_EEI_USAGE_MPLS_PORT` or encapsulation pointer
3. `info.l3a_intf` – Pointer to L3 egress (FEC)

5.6.3 Shell Commands

None

5.6.4 Application Reference

Usage example of EEI in the host table:

- Type – CINT reference
- Path – `src/examples/sand/cint_sand_service_using_eei.c`

5.7 InLIF EEI Construction

When using Cross-Connect to connect to an MPLS port (PWE), it may be useful to construct an EEI field to pass the MPLS-Label and push-profile information, which may optimize and skip the FEC table for MPLS-Port P2P objects.

By using an EEI field for Cross-Connect, the following limitation applies on the VLAN-Port:

When using EEI and two VLAN tags for VLAN-Translation, it is required to create the VLAN-Port with Protection or with the WIDE-DATA flag.

5.8 InLIF Profile in IOP Mode

InLIF profile in IOP mode is supported in a the same way as the normal mode. The difference is in the allocation of the egress profile bits. The maximum number of bits is two, as it was in the legacy devices. Supported applications are:

- Two bits orientation (network groups), configured in the same manner as in the BCM88690. In the BCM88690, this is supported by configuring the SOC property `in_lif_profile_egress_allocate_orientation` with the value 3. If the configuration of the SOC property is not explicitly defined, the behavior is the default one.
- Two bits of the same interface filter, if both device and system scope filtering is enabled. For BCM88690 the SOC property `in_lif_profile_egress_allocate_same_interface_mode` should be set to 3. The legacy device configuration is done using `bcm886xx_logical_interface_same_filter_enable` and `bcm886xx_logical_interface_bridge_filter_enable`.

  To support only system-scope filtering, assuming the device scope is enabled by default, configure the SOC property `in_lif_profile_egress_allocate_same_interface_mode = 1`. To allocate a LIF profile bit that supports only device scope filtering, use `in_lif_profile_egress_allocate_same_interface_mode = 2`.
- The allocation of LIF profile bits that support both orientation and same interface filtering is limited. Only 1 bit of orientation and 1 bit for same interface can be used. The configuration with SOC properties is as follows:
  - `in_lif_profile_egress_allocate_same_interface_mode = 1` and `in_lif_profile_egress_allocate_orientation = 2` for orientation with system scope
  - `in_lif_profile_egress_allocate_same_interface_mode = 2` and `in_lif_profile_egress_allocate_orientation = 2` for orientation with device scope

NOTE: In the BCM88830, the configuration limitation for the usage of the same interface with orientation does not apply. The configuration is limited only to mimic the legacy devices. In other words, it is possible to have 2 bits of orientation and 2 bits of the same interface together. The values of the SOC properties that configure this behavior are as described. For example, to configure 2 bits of orientation, which is the maximum, with a configurable device and system scope, the SOC properties are as follows:
`in_lif_profile_egress_allocate_same_interface_mode = 3` and
`in_lif_profile_egress_allocate_orientation = 3`.
The following configuration reserves 4 bits of the LIF profile for egress usage.

This behavior may be useful when a BCM88830 device connects to a device in the BCM88670 family directly with no BCM88690 or BCM88800. If the BCM88830 connects to a device in the BCM88670 family and a BCM88690 or BCM88800 device, the limitation of 2 bits must be applied to preserve the same allocation of bits over the system headers. In other words, the SOC property values must be aligned to all devices in the system.

5.8.1 SOC Properties

To configure the BCM88690 device, use the following SOC properties:

- `in_lif_profile_egress_allocate_orientation`
- `in_lif_profile_egress_allocate_same_interface_mode`

To configure the BCM88670 device, use the following SOC properties:

- `bcm886xx_logical_interface_same_filter_enable`
- `bcm886xx_logical_interface_bridge_filter_enable`
- `split_horizon_forwarding_groups_mode`

5.8.2 Configuration Flow

Configuration for orientation and same interface mode is the same as in non IOP mode. For more information, refer to Section 5.8, InLIF Profile in IOP Mode.

5.8.3 Shell Commands

None

5.8.4 Application Reference

Example configuration for orientation in IOP mode:

- Type – CINT reference
- Path – `src/examples/sand/cint_sand_multi_device_evpn.c`

5.9 L3 Egress Ingress-FEC Entry: OutLIF Pointer as EEI

If an application requires two pointers, the BCM88670 system headers mode requires one pointer to be provided as OutLIF and one pointer to be provided as EEI (with encapsulation pointer information). To support this, a new FEC format is introduced with the EEI format. This special format is used only in BCM88670 system headers mode where the EEI header is required to support two pointers.

In this case, it is possible to configure the following FEC information:

- Valid Destination
- Encapsulation-pointer (OutLIF) as EEI

For more information on the FEC object, see Section 22.8, L3 Egress Object: Ingress-FEC.

5.9.1 SOC Properties

None

5.9.2 Configuration Flow

Create FEC by calling API `bcm_l3_egress_create(unit, flags, egr, if_id)`

- `flags` – `BCM_L3_INGRESS_ONLY` (FEC creation)
- `egr.destination` – Destination of the FEC, can be physical destination.
- `egr.intf` – OutLIF pointer (LIF type only)
- `egr.flags` – `BCM_L3_ENCAP_SPACE_OPTIMIZED` - Once set, FEC will construct EEI format of OutLIF pointer (FEC format C).

5.9.3 Shell Commands

None

5.9.4 Application Reference

For an example of usage for FEC with EEI as OutLIF pointer:

- **Type**: CINT reference
- **Path**: `src/examples/sand/cint_mpls_encapsulation_basic.c`

5.10 Traffic Injecting

Each generation of a device has its own ingress header formats. Under IOP mode, the BCM88690 can use the BCM88690 format of ITMH and PTCH headers, or BCM88670 format of ITMH and PTCH headers. This is done per port basis. To summarize ITMH and PTCH formats are decided by the ingress device:

- If the ingress device is BCM88690, ITMH and PTCH headers are BCM88690 format or BCM88670 format per port decision
- If the ingress device is BCM88670, ITMH and PTCH headers are BCM88670 format per port decision

As for system headers that pass between Ingress to Egress (for example, FTMH, Internal Header, UDH and so on), on IOP mode, BCM88690 process the *BCM88670 system headers format only*. In that case, FTMH, UDH and Internal headers are process with BCM88670 formats only.

For traffic injection with ingress and system headers, BCM88690 will process the ingress headers (PTCH/ITMH) and on some cases system-headers like Internal-Header by the ingress PMF while system headers (for example FTMH and UDH) are treated as a payload transferring to egress in this case.

The following table summarizes the supported header formats.

Ingress Device	Type	Header Name	Supported Format (BCM88670/BCM88690)
BCM88670	Ingress	PTCH	BCM88670
BCM88670	Ingress	ITMH	BCM88670
BCM88670	System	FTMH	BCM88670
BCM88670	System	Internal header	BCM88670
BCM88670	System	UDH	BCM88670
BCM88690	Ingress	PTCH	BCM88690/BCM88670 (per port basis)
BCM88690	Ingress	ITMH	BCM88690/BCM88670 (per port basis)
BCM88690	System	FTMH	BCM88670
BCM88690	System	Internal header	BCM88670
BCM88690	System	UDH	BCM88670

When the BCM88690 incoming-port receives injected incoming traffic with BCM88670 (JR1) system headers mode (IOP mode) or BCM88690 (JR2) system headers, it depends on port type INJECTED_2_PP_JR1_MODE or INJETED_2_PP.

An exception is for OAMP, which always injects traffic with the BCM88670 (JR1) system header under IOP mode. That is, it must be set with the port header type INJECTED_2_PP_JR1_MODE.

For more information on port header's type, see Section 2.1, Packet Processing Ports.

5.11 Egress Initial NWK-QoS

The BCM88690 always includes a NWK-QoS field. In the egress, the initial NWK-QoS is the ingress NWK-QoS in a packet that is L2 or L3 forward. In BCM88670 IOP mode, If a packet is IPv4/6 or MPLS forward and DSCP-EXP exists in the system headers then the initial NWK-QoS is mapped 1:1 from FHEI.DSCP-EXP(8); if a packet is IPv4/6 or MPLS forward and DSCP-EXP does not exist in the system-headers, the initial NWK-QoS is mapped 1:1 from the packet header; otherwise, the initial NWK-QoS is according to Traffic-Class (TC).

No configuration is required when QoS mappings are done. Consider this as part of the egress QoS mappings.

5.11.1 SOC Properties

None

5.11.2 Configuration Flow

None

5.11.3 Shell Commands

None

5.11.4 Application Reference

None

5.12 Push-Profile Allocation

EEI can be used for MPLS header construction. In such cases, a push command is required with an MPLS push profile. For BCM88690, only the MPLS push command and the push profile are required. They are available under IOP mode.

When EEI is decoded as an MPLS push command, a three bits push profile is carried along with a 20 bits MPLS label. In the egress, the push profile is used to address the push command table, which is used to decide which relevant MPLS header property is used (TTL/EXP pipe or uniform, control word, and so on).

The push profile between the ingress and the egress devices is the most significant part of the synchronization. It is required to insure that when a push profile is encoded in the ingress, the egress MPLS header construction is expected.

To achieve this, `bcm_mpls_port_add()` is used to configure the push profile across multiple device:

1. Configure a push profile with the same push profile ID in the ingress device and the egress device.
2. Configure the application using the push profile.

NOTE: Only bridge and L3 services can use the EEI resolution of a push profile. MPLS LSR service cannot use this.

5.12.1 SOC Properties

None

5.12.2 Configuration Flow

Initiate a push profile by calling the API `bcm_mpls_port_add()(unit, mpls_port)`:

- flags2 – `BCM_MPLS_PORT2_TUNNEL_PUSH_INFO`, to indicate this is for push profile initiation.
- `egress_label`: MPLS header property
  - `egress_label.label` – MPLS Label value
  - `egress_label.egress_qos_model.egress_ttl` – TTL model
  - `egress_label.egress_qos_model.egress_qos` – QoS model
  - `egress_label.ttl` – TTL value for TTL pipe model
  - `egress_label.exp` – EX value for QoS pipe model
  - `egress_label.qos_map_id` – Remark profile for next layer remark
  - `egress_label.flags`:
    - `BCM_MPLS_EGRESS_LABEL_ENTROPY_ENABLE` – Entry Label needed
    - `BCM_MPLS_EGRESS_LABEL_ENTROPY_INDICATION_ENABLE` – ELI is needed
  - flags: `BCM_MPLS_PORT_CONTROL_WORD` – Adding a control word.
  - `mpls_port_id` – Push profile ID. Encode as gport type MPLS-port and subtype `BCM_GPORT_SUB_TYPE_MPLS_PUSHe_PROFILE`

5.12.3 Shell Commands

None

5.12.4 Application Reference

Usage example of a service using EEI as an MPLS push profile:

- Type – CINT reference
- Path – `src/examples/sand/cint_sand_service_using_eei.c`

5.13 L2 Learning

5.13.1 DSP Encoding

When sending DSP packets to manage the MAC table, the following packet encodings might be used:

- BCM88650 (Arad) encoding
- BCM88670 (Jericho) encoding
- BCM88690 (Jericho2) encoding

The BCM88690 device works in BCM88690 (Jericho2) DSP encoding only.

The BCM88670 device works in Arad DSP encoding.

When the BCM88690 device is working with the BCM88670 in the same system, the BCM88690 device is expected to send DSP packets to the BCM88670 device using Arad encoding and receive DSP packets from the BCM88670 with Arad encoding.

This means BCM88670 (Jericho) devices create and consume DSPs where the DSP.Payload is encoded in Arad destination encoding.

For the BCM88690 only, when working in interop mode, the device creates and consumes DSPs, where the DSP.Payload is encoded in BCM88670 (Jericho) destination encoding. For more information, refer to EID#9094 in the *Device Errata* (88690-ER1xx).

A application exists in the reference code that re-encodes the DSP packet payload originating from BCM88670 (Jericho) destination encoding to Arad destination encoding. DSP packets received from other devices in the system are re-encoded from Arad destination encoding to BCM88670 destination encoding. The application involves a two-pass packet walk for DSPs generated from a BCM88690 device

5.13.2 MAC Limit – DSP Packets

The BCM88690-BX device cannot manage the MAC table by sending transplanted, deleted, or refreshed DSP packets when entries inside the MAC table reach the MAC limit. For additional information, refer to errata EID#9087 in the *Device Errata* (88690-ER1xx).

A workaround is implemented in the application reference. The workaround attempts to re-encode the DSP command to the refresh command.

5.13.3 Refresh DSP Packets – Key Field

The 8 most significant bits (MSBs) of a key refresh DSP packet are wrong. If a packet arrives at the egress of a Jericho device, it will generate a new MACT entry because of the wrong key. For further information, refer to errata EID#9133 in the *Device Errata* (88690-ER1xx). A workaround is introduced to clear these bits at OLP TX direction.

5.13.4 L2 Learning Application Summary

The following table provides a summary of the applications that must be installed for DSP packets in interop-mode.

Application	BCM88690-A0	BCM88690-BX	BCM88800	BCM88830
DSP Encoding	+	+	—	—
MAC limit – DSP packets	—	+	—	—
Refresh DSP packets – key field	+	+	+	+

For every DSP packet, different processes may be done according to the payload type and command type. The following list is examples of those workarounds:

- If the DSP payload destination format is FEC and the DSP command type is *not* insert, do not apply any workaround.
- If the DSP payload destination format is dst-sys-port and the DSP command type is *not* insert, apply the first workaround.
- If the DSP payload destination format is FEC and the DSP command type is insert, apply the second workaround.
- If the DSP payload destination format is dst-sys-port and the DSP command type is insert, apply both workarounds.
- For a DSP refresh, the 8 MSBs of the key are reset.

To implement those applications, different OLP ports and recycle ports are required. The following table summarizes the number of ports per device:

Port Type	BCM88690-A0	BCM88690-BX	BCM88800	BCM88830
OLP-ports	2 (240, 241)	4 (240-243)	1 (240)	1 (240)
RCY-ports	2	2	1	1

To support multiple DSP ports, the ports must be channelized, and DSP packets are redirected to different OLP channels where different processes are performed.

NOTE: The workarounds provided in the SDK assume that the `header_type_in` value of the OLP ports is INJECTED_2, which means that it uses ITMH in BCM88690 (Jericho2) format.

In addition, the application step allowing the field processor to support BCM88690 (Jericho2) ITMH must be enabled by setting the `appl_enable_field_itmh=1` SOC (for more information, see Section 11.26.1.1, ITMH).

Recycle ports redirect the DSP message to the second pass for destination decode converting. In the application reference example (see Section 5.13.8, Application Reference), recycle ports 41 and 44 can be used in field workaround code.

5.13.5 SOC Properties

For the BCM88690-A0, use the following SOC properties:

- `ucode_port_41.BCM8869X_A0=RCY.0:core_0.41`
- `ucode_port_44.BCM8869X_A0=RCY.3:core_0.44`
- `tm_port_header_type_in_41.BCM8869X_A0=ETH`
- `tm_port_header_type_in_44.BCM8869X_A0=ETH`
- `tm_port_header_type_out_41.BCM8869X_A0=ETH`
- `tm_port_header_type_out_44.BCM8869X_A0=ETH`
- `ucode_port_240.BCM8869X_A0=OLP.0:core_0.240`
- `ucode_port_241.BCM8869X_A0=OLP.1:core_0.241`
- `tm_port_header_type_in_240.BCM8869X_A0=INJECTED_2`
- `tm_port_header_type_in_241.BCM8869X_A0=INJECTED_2`
- `tm_port_header_type_out_240.BCM8869X_A0=ETH`
- `tm_port_header_type_out_241.BCM8869X_A0=ETH`

For the BCM88690-BX, use the following SOC properties:

- `ucode_port_41.BCM8869X_B0=RCY.0:core_0.41`
- `ucode_port_44.BCM8869X_B0=RCY.3:core_0.44`
- `tm_port_header_type_in_41.BCM8869X_B0=ETH`
- `tm_port_header_type_in_44.BCM8869X_B0=ETH`
- `tm_port_header_type_out_41.BCM8869X_B0=ETH`
- `tm_port_header_type_out_44.BCM8869X_B0=ETH`
- `ucode_port_240.BCM8869X_B0=OLP.0:core_0.240`
- `ucode_port_241.BCM8869X_B0=OLP.1:core_0.241`
- `ucode_port_242.BCM8869X_B0=OLP.2:core_0.242`
- `ucode_port_243.BCM8869X_B0=OLP.3:core_0.243`
- `tm_port_header_type_in_240.BCM8869X_B0=INJECTED_2`
- `tm_port_header_type_in_241.BCM8869X_B0=INJECTED_2`
- `tm_port_header_type_out_240.BCM8869X_B0=ETH`
- `tm_port_header_type_out_241.BCM8869X_B0=ETH`
- `tm_port_header_type_in_242.BCM8869X_B0=INJECTED_2`
- `tm_port_header_type_in_243.BCM8869X_B0=INJECTED_2`
- `tm_port_header_type_out_242.BCM8869X_B0=ETH`
- `tm_port_header_type_out_243.BCM8869X_B0=ETH`

For the BCM88800, use the following SOC properties:

- `ucode_port_41.BCM8880X=RCY.0:core_0.41`
- `tm_port_header_type_in_41.BCM8880X=ETH`
- `tm_port_header_type_out_41.BCM8880X=ETH`
- `ucode_port_240.BCM8880X=OLP.0:core_0.240`
- `tm_port_header_type_in_240.BCM8880X=INJECTED_2`
- `tm_port_header_type_out_240.BCM8880X=ETH`

The BCM88830 uses the same properties as the BCM88800.

5.13.6 Configuration Flow

Configure recycle port for two-pass solution:

```
bcm_port_control_set(unit, recycle_port, bcmPortControlDistributeLearnMessageMode, TRUE)
```

NOTE: This is required only for the BCM88690-BX device.

For configurations required for the Field Processor, see Section 5.13.8, Application Reference.

5.13.7 Shell Commands

None

5.13.8 Application Reference

For an example of an implemented workaround for L2 learning in interop mode, refer to the following:

- Type: Application reference
- Path: `src/appl/reference/dnx/appl_ref_field_wa_init.c`

5.14 EVPN Application

The following changes exist when working in BCM88670 system headers mode (IOP mode) in the EVPN application:

- Field processor for ESI filtering – See Section 35.12, Field Processor for ESI Filtering in BCM88670 (IOP) Mode.
- ESI encapsulation – See Section 35.5, ESI Encapsulation in IOP Mode.
- ESI label copying – Required Field-Processor application; see Field Processor for ESI label copying in IOP mode.

5.15 MPLS PHP

In legacy devices, when PHP is performed, identifying the upper-layer-protocol required the Field-Processor application. At the ingress, a field group is created to check the forwarding-label BoS bit, and it updates the EEI or EEDB pointer to select the correct upper-layer-protocol.

In BCM88690 family devices, the same field-group is required, and there is an additional requirement: the OutLIF EEDB with PHP operation entry must be in the dedicated range of 0x3EF80 to 0x3EFFF. This required range is validated by the SDK. When including the WITH_ID allocation, the SDK verifies the ID is in the range. When WITH_ID is not included, the SDK allocates from that range.

NOTE: When the ingress device is in the BCM88690 family, the OutLIF EEDB with PHP operation can act only as a *pure PHP* rule. The other functions in EEDB (statistic, QoS, MTU profile, and so on) are not functional.

5.16 OAM Egress Snooping

When working in IOP mode, the same interface filter is enabled for egress snoop packets.

If the destination port of the snoop packet is the same as the source system port in the second pass, the snoop packet will be dropped.

The source system port of the egress snoop packet can be set using the `bcm_mirror_header_info_set` API.

5.16.1 SOC Properties

None

5.16.2 Configuration Flow

None

5.16.3 Shell Commands

None

5.16.4 Application Reference

Example of disabling the same interface filter on a port:

- Type: CINT reference
- Path: `src/examples/dnx/oam/cint_oam_basic.c`

5.17 Fast Flush for Ring Protection

Fast flush is supported in IOP mode differently than in native mode. In native mode, it is performed by creating an AC with a group. This group is learned and stored as a field of an L2 entry and can flush MACT entries that share the same group. For more information, see Chapter 37, ITU-T G.8032 Ring Automatic Protection Switching.

In IOP mode, ACs are grouped when they have a mutual FEC that acts as a BCM88690 (Jericho2) group, similar to what is done in the BCM88670 family devices. The learned L2 entries of these ACs have the same FEC ID as a destination. The flush operation defines the FEC as the flush matching key, masking all other key fields, thus flushing all MACT entries associated with devices on the ring.

5.17.1 SOC Properties

None

5.17.2 Configuration Flow

1. Create a FEC by calling `bcm_l3_egress_create(unit, flags, &egr, &fec_id)`:
   - `flags`: `BCM_L3_INGRESS_ONLY`
   - `egr.port`: The destination could be a physical port.
   - `egr.intf`: set it as 0.
2. Create ACs and associate each AC to a FEC by calling `bcm_vlan_port_create(unit, vlan_port)`
   - `vlan_port.failover_port_id`: FEC gport, `BCM_L3_ITF_FEC_TO_GPORT_FORWARD_GROUP(fec_gport, fec_id)`;
   - `vlan_port.group`: 0 -- Means create an AC learning information that is FEC only; otherwise, means create an AC learning information that is FEC+outlif as required for Ring Protection.
3. Flush MACT DB by FEC ID by calling `bcm_l2_replace_match(unit, flags, match_addr, mask_addr, NULL, NULL)`:
   - `flags`:
     - `BCM_L2_REPLACE_DELETE`
     - `BCM_L2_REPLACE_NO_CALLBACKS`
     - `BCM_L2_REPLACE_IGNORE_DES_HIT`
     - `BCM_L2_REPLACE_MATCH_STATIC | BCM_L2_REPLACE_MATCH_DEST`
     - `BCM_L2_REPLACE_MATCH_GROUP`
   - `match_addr.port`: Any of the grouped AC's gport
   - `mask_addr.group`: Group is 0 and flags is `BCM_L2_REPLACE_MATCH_GROUP`, to mask the OutLIF field and destination filter is FEC ID only.

5.17.3 Shell Commands

None

5.17.4 Application Reference

Example of usage for fast flush when DNX works at JR1 mode

- **Type:** CINT reference
- **Path:** `src/examples/dnx/cint_ring_protection.c`

Chapter 6: Modular Database Profile

6.1 Introduction

The Modular Database (MDB) profile determines the distribution of the MDB hardware resources across all MDB tables. The profile determines how many entries each table can accommodate. Depending on the profile, a certain table can support a large number of entries or be completely disabled.

Several MDB profiles are provided with the SDK, as defined by the PP architecture. To select the MDB profile that best meets the application requirements, use the MDB profile SoC property. The entire MDB configuration is determined solely by the MDB profile SoC property. Switching between different profiles does not require any compilation changes.

During the SDK initialization, the MDB init phase configures and associates the MDB hardware resources with the MDB tables as defined in the MDB profile. During runtime, the different entry operations are directed to the appropriate hardware resources within the lower layer of the SDK.

6.2 SOC Properties

The following SoC properties apply:

- `mdb_profile` – The MDB profile controls the allocation of hardware resources to different tables.

6.3 MDB Batch Optimization

To enhance the entry insertion and deletion performance in the MDB, it is possible to automatically accumulate multiple entry changes in cache and periodically commit them as a batch.

This performance enhancement turns relevant BCM APIs to asynchronous, meaning the BCM API will return success before the MDB entry is committed to HW. To ensure all pending HW changes are committed, use the following BCM API:

```
bcm_switch_control_set(unit, bcmSwitchDescCommit, 1);
```

NOTE: This feature assumes MDB caching is enabled, which it is by default in the SDK.

6.3.1 SOC Properties

The descriptor DMA SOC properties must be nonzero:

- `dma_desc_aggregator_chain_length_max=500` – Maximum number of descriptors in a single chain.
- `dma_desc_aggregator_buff_size_kb=100` – Total size in KB of DMA-able memory allocated in favor of the descriptor DMA double-buffer.
- `dma_desc_aggregator_timeout_usec=1000` – Timeout between the creation of a descriptor chain and its commit to HW.

NOTE: `table_dma_enable` and `tslam_dma_enable` must be enabled to enable descriptor DMA.

6.3.2 Configuration Flow

None (performed on SDK initialization according to the SOC properties)

6.3.3 Shell Commands

The shell commands are as follows:

- `DmaDescAgg STATS` – Prints various statistics and information regarding the committed descriptor chains. Can be cleared. May help in determining the optimal SOC properties for a specific application.
- `DmaDescAgg ENAble` – Enable or disable the descriptor DMA.

Chapter 7: Logical Interface (LIF) Management

7.1 Introduction

Logical interfaces (LIFs) emulate L2 tunnels (for example, AC, PWE, and VXLAN) or L3 tunnels (for example, MPLS/IP-Tunnels) and have incoming and/or outgoing interfaces. A logical tunnel ID is known as the *Global-LIF-ID*. For the incoming interface, the ID is the *In-Global-LIF*, and for the outgoing interface the ID is the *Out-Global-LIF*.

The Global-LIF-ID is used as the software object handling ID as well as the hardware-representation over the system-headers and the different forwarding/learning/ACL databases. The software object handling ID is used as an input to configure the BCM LIF APIs and to wrap with the GPORT entity. For more information, see Chapter 3, L2 and L3 Ports (GPORTs). For the hardware-representer, the object-ID is used to point to L2/L3 tunnels from the forwarding tables and can be used as part of the learning information and for same-interface filtering. In addition, the object-ID can be used by the ACL databases (Field Processor APIs), multicast APIs.

A logical interface can be considered local only to a specific device (a device-scope LIF) or to the entire chassis/system (system-scope LIF). For a device-scope LIF (for example, an AC that is not located over a trunk and it is not protected), its interface is known only to the local-device, and its unique ID is a mapping of {FAP-ID, Global-LIF-ID}. For a system-scope LIF (for example, PWE, interface over trunk), its unique ID is just the Global-LIF-ID.

If a Global-LIF-ID has a local meaning for a specific device, its hardware entity is being allocated. For In-Global-LIF, it is located at the InLIF tables and indexed by In-Local-LIF. For Out-Global-LIF, it is located and indexed by Out-Local-LIF. A Global-LIF-ID can have several hardware entries, depending on its required size. In addition, the InLIF table is used to map In-Local-LIF to its In-Global-LIF, and a table is introduced to map an Out-Global-LIF to its Out-Local-LIF.

The Local-LIF ID is restricted to a specific device and the SDK manages it internally by its allocation algorithms. The allocated IDs cannot be controlled by the user and are not exposed. However, some properties of the allocation algorithms can be tuned/optimized according to user knowledge of allocated interface types:

- The global amount of entries to allocate are decided according to MDB profile. For more information, see Chapter 6, Modular Database Profile.
- Tune information according to interface types in EEDB. For more information, see Section 7.4, EEDB Management.

Unlike the Local-LIF ID, the Global-LIF-ID is a logical number that is not dependent on hardware restrictions. The Global-LIF-ID also has system-scope aspects. As such, the user application must handle the Global-LIF-ID allocation as part of the BCM LIF APIs. The SDK provides two main methods to work with Global-LIF-ID:

1. When a LIF object is created on certain device and the `_WITH_ID` flag is not set, the SDK returns a free Global-LIF-ID that is used as a handle to the object in all subsequent references for the local-device.
2. When a LIF object is created on certain device and the `_WITH_ID` flag is set, the SDK makes sure the Global-LIF-ID applies to Global-LIF-ID ranges, makes sure it is available, and returns back a handle to the object in all subsequent references for the local-device.

The BCM API calls operate strictly within the context of a single device, and so does the allocation manager of Global-LIF-ID. However, with synchronized system objects, an object must have the same ID in all devices. Since the allocation manager is strictly local, method 1 (`_WITH_ID` flag is not set) is useful when there is only one device in the system. But with a system/chassis, method 2 (`_WITH_ID` flag is set) is required in order to sync between all devices to allocate the same object-ID between all required devices.

Method 2 is the recommended flow of work (and required for system/chassis applications). In this case, it is important to understand the Global-LIF-ID allocation restrictions. For more information on Global-LIF-ID allocation restrictions, see Section 7.3, Global LIF-ID Management.

7.2 Application Configuration Checklist

- Global-LIF ID Management
- EEDB Management
- LIF-profile Resources

7.3 Global LIF-ID Management

The Global-LIF-ID shares the same namespace as the ETH-RIF ID. Due to this, and based on the number of supported ETH-RIFs, the ETH-RIF IDs are set to number between <0–#nof_rifs–1>, and Global-LIF-IDs start from nof_rifs. For more information on ETH-RIF-ID, see Section 22.6, ETH-RIF (L3 VSI Interface) and VRF Definition.

The following table lists the LIF-API and their Global-LIF allocation flags.

API Name	Configuration Flag	LIF Type	In/Out/ Sy	Object-ID (Wrap + Gport Notation)
bcm_l3_egress_create	BCM_L3_EGRESS_ONLY BCM_L3_WITH_ID	ARP	Out	encap_id
bcm_vlan_port_create	BCM_VLAN_PORT_WITH_ID vsi = 0	Service AC-LIF	Sy	vlan_port_id
bcm_vlan_port_create	BCM_VLAN_PORT_WITH_ID BCM_VLAN_PORT_INGRESS_ONLY vsi != 0	VLAN-Translation In-AC	In	vlan_port_id
bcm_vlan_port_create	BCM_VLAN_PORT_WITH_ID BCM_VLAN_PORT_EGRESS_ONLY vsi != 0	VLAN-Translation Out-AC	Out	vlan_port_id
bcm_mpls_port_add	BCM_MPLS_PORT_WITH_ID BCM_MPLS_PORT2_INGRESS_ONLY	In-PWE	In	mpls_port_id
bcm_mpls_port_add	BCM_MPLS_PORT_WITH_ID BCM_MPLS_PORT2_EGRESS_ONLY	Out-PWE	Out	mpls_port_id
bcm_mpls_tunnel_switch_create	BCM_MPLS_SWITCH_WITH_ID	In-MPLS-Tunnel	In	tunnel_id
bcm_mpls_tunnel_initiator_create	BCM_MPLS_EGRESS_LABEL_WITH_ID	Out-MPLS-Tunnel	Out	tunnel_id
bcm_tunnel_terminator_create	BCM_TUNNEL_WITH_ID	In-IPv4/6 Tunnel	In	tunnel_id
bcm_tunnel_initator_create	BCM_TUNNEL_WITH_ID	Out-IPv4/6 Tunnel	Out	tunnel_id

7.3.1 Shell Commands

```
gpm hw gport=<gport>
```

- Since Global-LIF is mostly indicated by a gport, it's possible to get the Global-LIF information (mostly local LIF mapping) from the gport to hardware resources diagnostic.
- Output: Global-LIF for the gport, ingress/egress Local-LIF for the gport, and DBAL table dump for the local LIFs (only the ones relevant for the gport).

7.4 EEDB Management

EEDB management is split into two parts:

- Part I: Introduce the Logical-Phase and how different encapsulation objects relate to the Logical-Phase.
- Part II: EEDB entry allocation and user knowledge.

Each EEDB entry is associated with a logical phase of value 1 to 8. The EEDB can be accessed only once per stage, at most eight times per packet. The encapsulation entry phases must be ordered according to the order of the header encapsulation on the packet. For example, if PWE and MPLS encapsulation tunnels are processed for a certain packet, the first tunnel PWE, (position to BOS) must be in an access phase lower than the MPLS tunnel.

In some BCM APIs, the phase is fixed. In other BCM APIs, the phase is chosen automatically but can be configured in a certain range using the following property:

```
/* Encapsulation Access stage */
typedef enum bcm_encap_access_e {
    bcmEncapAccessInvalid = 0,      /* INVALID (automatic decided) */
    bcmEncapAccessTunnel1 = 1,      /* FIRST_ACCESS */
    bcmEncapAccessTunnel2 = 2,      /* SECOND_ACCESS */
    bcmEncapAccessTunnel3 = 3,      /* THIRD_ACCESS */
    bcmEncapAccessTunnel4 = 4,      /* FOURTH_ACCESS */
    bcmEncapAccessNativeArp = 5,    /* NATIVE ARP ACCESS */
    bcmEncapAccessNativeAc = 6,     /* NATIVE AC ACCESS */
    bcmEncapAccessArp = 7           /* OUTER ARP ACCESS */
} bcm_encap_access_t;
```

Each enum represents the following phases:

Enum Type	Logical-Phase (1 to 8)
bcmEncapAccessRif	1
bcmEncapAccessNativeArp	2
bcmEncapAccessNativeAc	3
bcmEncapAccessTunnel1	3
bcmEncapAccessTunnel2	4
bcmEncapAccessTunnel3	5
bcmEncapAccessTunnel4	6
bcmEncapAccessArp	7
bcmEncapAccessAc	8

Notes:

- Logical-Phase 1 is reserved for RIF.
- Logical-Phase 8 is reserved for AC and ARP+AC.

The enum value bcmEncapAccessInvalid is used by some APIs to select a default phase. The retrieved value in the respective APIs will be bcmEncapAccessInvalid whenever the default value is used, regardless of which value was used to configure it (bcmEncapAccessInvalid or the default value explicitly).

The following table summarizes some of the LIF APIs and the phases they configure.

Table 1: LIF APIs and the Phases They Configure

API Name	Configuration Flag	LIF Type	Default Phase	Possible Phases
bcm_l3_intf_create	—	RIF	1	1
bcm_l3_egress_create	BCM_L3_EGRESS_ONLY	ARP	7	7, 8
	BCM_L3_EGRESS_ONLY BCM_L3_FLAGS2_VLAN_TRANSLATION	ARP + AC	8	8
	BCM_L3_EGRESS_ONLY BCM_L3_NATIVE_ENCAP	NATIVE ARP	2	2
bcm_vlan_port_create	—	AC	8	8
	BCM_VLAN_PORT_NATIVE	NATIVE AC	3	3
bcm_mpls_port_add	BCM_MPLS_PORT2_EGRESS_ONLY	MPLS PORT	3	1 to 4
bcm_mpls_tunnel_initiator_create	—	MPLS TUNNEL	4	1 to 7
bcm_tunnel_initiator_create	—	IPv4/6 TUNNEL	3	3 to 6 (IPv4) 4 to 5 (IPv6)

NOTE: For an MPLS tunnel, Logical-Phase 7 can be used only if ARP+AC is used.

It is possible for the user to control the size of each EEDB logical-phase and its data granularity. Sizes are represented by XL, L1, L2, M1, M2, M3, S1, S2. The decision depends on user knowledge of its applications. For more information, see the SOC property information in the next section.

Other applications can also allocate entries in the EEDB.

7.4.1 SOC Properties

- Map every logical phase (the order in which the OutLIF is accessed) to its physical phase (the databases it can access). The logical phases are 1 to 8, and the physical phases are represented by their size:
  - XL: Has the largest MDB cluster and can use all the dedicated EEDB banks to store data.
  - L1: Has the second largest MDB cluster, and can access half the dedicated EEDB banks to store data.
  - L2: Has the third largest MDB cluster, and can access half the dedicated EEDB banks to store data.
  - M1: Has the fourth largest MDB cluster, and can access a quarter of the dedicated EEDB banks to store data.
  - M2: shares the fourth largest MDB cluster with M1, and can access a quarter of the dedicated EEDB banks to store data.
  - M3: Shares the third largest MDB cluster with L2, and can access a quarter of the dedicated EEDB banks to store data.
  - S1: Shares the second largest MDB cluster with L1, and can access an eighth of the dedicated EEDB banks to store data.
  - S2: Shares the largest MDB cluster with XL, and can access an eighth of the dedicated EEDB banks to store data.

  `outlif_logical_to_physical_phase_map_<phase_number>=<Type>`.

7.4.2 Configuration Flow

None

7.4.3 Shell Commands

Local OutLIF MDB management through SW diagnostics:

- `lif localoutlif dump` – Show information on all logical phases OutLIF banks usage.
- `lif localoutlif phase dump` – Show information on local OutLIF banks per given logical phase.

7.4.4 Application Reference

None

7.4.5 API Descriptions

None

7.5 LIF Profile Resources

LIF profile is a LIF property that exists in the InLIF (also known as InLIF profile), OutLIF (OutLIF ERPP profile and OutLIF ETPP profile) and used by the data-plane properties as well as the field processor. With the field processor it can be used as a preselector, qualifier or action. With the data-plane it is also used by some applications as additional LIF information for context selection and programmable stages as well as hard logic decisions. In addition, part of the LIF-profile information is being transferred from ingress to egress on top of system headers to allow data-plane hard-logic decisions and ACL rules at the egress.

7.5.1 InLIF Profile

For InLIF profile, the allocation algorithm splits the resource into three different sub-resources:t

- PMF sub-resources. For this purpose, the SOC properties are used and indicate how many bits to allocate. For the In-LIF profile, the amount of bits is allocated with `pmf_in_lif_profile_nof_bits`. For the ETH RIF profile, the relevant SOC property is `pmf_in_rif_profile_nof_bits`.
- Egress sub-resources. InLIF profile properties that are being transferred from ingress to egress on top of system headers. In such cases, the algorithm allocates in advance the amount of bits and synchronizes the values across all devices in the system. SOC properties with prefix `in_lif_profile_egress_allocate_xxx` are used to set the amount of values (amount of combinations) required for each application. Given this information, the allocation algorithm determines the number of bits and their values over the system-headers.
- Ingress sub-resources. The remaining resources are allocated for InLIF profile properties that are used by the data-plane ingress decisions. They are dynamically allocated upon LIF API request.

The following applications are using InLIF profile resources:

Application	Sub-Resource	API/SOC Property
Private-Public-Optimize	Ingress	`bcm_l3_ingress_create`
RPF-Mode	Ingress	`bcm_l3_ingress_create`
Routing-Enablers	Ingress	`bcm_l3_ingress_create`
Orientation/Network groups	Egress	Different LIF APIs (VLAN-Port, MPLS-Port, VXLAN-Port), `bcm_port_class_set XXX` SOC property: `in_lif_profile_allocate_orientaion` For more details refer to Section 20.2, Split Horizon (Network Groups)
Same-Interface filtering	Egress	`bcm_port_control_set` with type `bcmPortControlBridge` for device scope filtering and `bcmPortControlLogicalInterfaceSameFilter` for system scope filtering. The SOC property to control the allocated values for the same interface bits in the egress is `in_lif_profile_egress_allocate_same_interface_mode`.
RX-trap protocol profile	Ingress	`bcm_rx_trap_protocol_profiles_set`
OAM-profile	Ingress	OAM profile can be modified for LIF and RIF. The LIF OAM profile is supported using the `bcm_port_control_set` API with the type `bcmPortControlOamDefaultProfile`. To assign an OAM profile property on an existing RIF use the `bcm_ingress_create` API with the `oam_default_profile` field, part of the `bcm_l3_ingress_t` structure
IVE PCP mapping	Egress	`bcm_port_control_set` with the type `bcmPortControlOuterPolicerRemark` for IVE outer PCP mapping according DP profile and `bcm_port_control_set` with the type `bcmPortControlInnerPolicerRemark` for IVE inner PCP mapping according DP profile. The SOC property to control the allocated values for DP profile in the egress is: ■ `in_lif_profile_egress_allocate_policer_outer_dp` ■ `in_lif_profile_egress_allocate_policer_inner_dp`

7.5.2 SOC Properties

- `in_lif_profile_egress_allocate_orientation` – Configure the number of supported values that can be allocated for orientation
- `in_lif_profile_egress_allocate_policer_inner_dp` – Configure the number of supported values that can be allocated for inner DP
- `in_lif_profile_egress_allocate_policer_outer_dp` – Configure the number of supported values that can be allocated for outer DP
- `in_lif_profile_egress_allocate_same_interface_mode` – Configure the number of supported values that can be allocated for the same interface. The supported modes are as follows:
  - Device-Scope (Check In-Port equals Out-Port and InLIF equals OutLIF)
  - System-Scope (Check only InLIF equals OutLIF)
  - Disable (Never check).

7.5.3 OutLIF ERPP Profile

OutLIF ERPP profile can be used by the egress PMF for ACL rules on the OutLIF.

To set it, call `bcm_port_class_set(unit, intf_gport, bcmPortClassFieldEgressVport, profile_value);`

where:

- `intf_gport` – OutLIF gport
- `profile_value`

NOTE: If the interface is a RIF, it should be first converted to tunnel gport using `BCM_GPORT_TUNNEL_ID_SET` macro.

7.5.4 OutLIF ETPP Profile

The following applications are using an OutLIF:

- OutLIF Split-Horizon – LIF profile is mapped to an out network group. The filtering decision is done according to both ingress and egress groups and filter configuration. For more details refer to Section 20.2, Split Horizon (Network Groups).
- OAM PCP mapping – LIF profile is mapped to OAM PCP based counting profile. The profile is configured using the `bcm_qos_port_map_set` API. For more information refer to Chapter 10, QoS and Chapter 33, Operations, Administration, and Maintenance.

Chapter 8: L2 VPN (VSI) Object Management

Various applications use the L2 VPN object. In DNX architecture, the L2 VPN is usually referred to as the *VSI*. In the SDK, the typedef `bcm_vpn_t` is often used to indicate the VPN field. Some cases use other names, like `bcm_vlan_t`.

Each application that uses a L2 VPN has a different software namespace for it (for example, Vswitch VPN, MPLS-Port VPN, or VXLAN VPN), while in hardware, the VSI-ID has the same meaning for all applications. For each VSI, the SDK stores an indication of which application created it, and multiple applications can create the same VSI. In hardware the fields overlap, and so is the SDK configuration. The VSI create and destroy actions are done per application-type. For example, when an MPLS VPN is used, the SDK also allows the creation of a Vswitch VPN. Still, the configuration in hardware is applied according to the last API called and not per application-type.

VSI and ETH-RIF share the same HW table at the ingress. Because of this, some of the fields are shared between L2 and L3 traffic.

The APIs listed in the following table allocate VSI-VPN.

VPN Type	BCM APIs
Vswitch	`bcm_vlan_create, bcm_vswitch_create, bcm_vswitch_create_with_id`
MPLS	`bcm_mpls_vpn_id_create`
VXLAN	`bcm_vxlan_vpn_create`

In addition, it is expected to create the VSI/VPN before using VSI control APIs. The VSI control APIs are listed in the following table.

API Name	Description
`bcm_vlan_control_vlan_set`	Control additional VSI properties, for example, L2 and Stat-related properties. It also sets PMF bits (see Section 11.24, LIF and Port Profiles).
`bcm_vrrp_config_add, bcm_l2_station_add`	Configure VRRP and Multiple-My-MAC enhancements.
`bcm_stg_vlan_add`	Configuration for STG.

NOTE: The VSI size is up to 16 bits.

For more information about API usage and descriptions, see Chapter 21, Ethernet Bridge, and Chapter 22, IP Router.

Chapter 9: FEC and ECMP Management

9.1 FEC Management

Some FEC entries can be defined as protection-pair entries (also known as *Super-FEC*), which use two entries of FEC or an unprotected single-entity FEC. It is possible to point from the FEC at a lower hierarchical-level to a higher-level, but not the opposite (higher-to-lower). Forwarding tables (or Cross-Connect objects) can point to any FEC at any hierarchical-level.

The FEC management must decide the FEC-IDs within predefined FEC ID ranges. The MDB profile determines the scale of each FEC hierarchy. The SDK determines the FEC-ID ranges at initialization time.

It is, however, possible to determine the FEC hierarchies allocation order using the `fec_id_range_position_x` SOC property.

To get the FEC ranges and their starting address, use the `bcm_switch_fec_property_get` API. For more information about getting the FEC ranges, see Section 22.8, L3 Egress Object: Ingress-FEC.

The SDK allows the usage of internal FEC management, where the FEC-IDs are decided by the SDK and provided back to the user. This approach is also known as *Without-ID*. When working with the *Without-ID* approach and a consecutive range of available FEC IDs is needed (mainly for creating ECMP groups), the `bcm_l3_egress_allocation_get` API will find such a range if there is one. Another approach is to have external FEC management. In this approach, known as *With-ID*, the user acquires the legal range of each hierarchy and provides an FEC-ID within that range when creating an FEC object. In this case, the SDK validates the provided FEC-ID is legal.

Working with the external FEC management approach involves the following considerations:

- If a protection-pair is required, the members of the protection-pair are any two consecutive FECs where the first ID is an even number, and the second ID is an odd number.
- The range of FECs between 0 to *Number-of-ECMP* is reserved for signaling ECMP resolution only. Still, it is possible to allocate FEC entries in the overlapping ECMP range. An FEC is allocated into that range only if the `BCM_L3_WITH_ID` flag and the `BCM_L3_FLAGS2_ECMP_RANGE_OVERLAP` flag are present (both flags are used in the `bcm_l3_egress_create` API).

NOTE: Each FEC allocated in the shared range with ECMP must be used as part of an ECMP group and not pointed directly from other forwarding or ACL databases.

The creation of an FEC and providing its ID is done as part of `bcm_l3_egress_create` API. For more information, see Section 22.8, L3 Egress Object: Ingress-FEC.

9.2 ECMP Management

The FEC members inside an ECMP group must have an identical protection state (all FECs inside an ECMP group must be protected, or none are protected).

In general, all the FEC members of an ECMP group must have consecutive IDs. However if the ECMP members are protected, only the even FEC ID members are added to the group, and the consecutive IDs are in gaps of two.

Chapter 10: QoS

10.1 Introduction

The Quality of Service (QoS) module combines the following applications:

- Per-Hop Behavior (PHB)
- Remarking/Marking
- ECN
- TTL

Most of the applications share similar hardware concepts and similar BCM API sequences. The configuration for QoS applications is usually per QoS type (PHB, Remarking, ECN, and TTL) and packet type (L2, IPv4, IPv6, MPLS, and so on).

This section provides a general overview of how to use and configure the QoS module.

10.2 Application Configuration Checklist

The following QoS applications exist in the SDK and must be configured:

- Ingress PHB/remarking
- Ingress VLAN translation VLAN-PCP QoS
- Ingress VLAN translation Policer QoS
- Ingress ECN
- Ingress TTL
- Egress policer QoS
- Egress OAM QoS
- Egress marking/remarking
- Egress ECN
- Egress TTL
- Egress MPLS PHP
- Egress VLAN translation VLAN-PCP QoS
- Egress CoS remapping

10.3 Ingress PHB/Remarking

Per Hop Behavior (PHB) allows the traffic class (TC) and drop precedence (DP) to be assigned according to packet properties. Through a similar method, Remark (RM) allows the normalization of a number that represents the packet's QoS called *Nwk-QOS*. Nwk-QOS is later used by the egress-device for stamping appropriate QoS fields in the outgoing header.

PHB and Remark methods have the same packet-flow. Incoming packets pass through several incoming interfaces (Incoming-Port, InLIFs, VSI, and In-RIFs), where each interface can assign different QoS properties:

- QoS-Layer-Index
- QoS-Profile

The QoS-Model procedure determines which interface influences the QoS properties. The following QoS-Model options might be available for the incoming interface:

- Uniform
- Pipe
- Short-pipe
- Stuck

The following table summarizes the decision of QoS properties given different incoming interface QoS models when X + 1 is the forward layer.

Table 2: Incoming Interfaces and QoS Property Decisions

Incoming-Interface (X)				
Incoming-Interface(X – 1)		**Uniform(X)**	**Pipe(X)**	**Short-Pipe(X)**
Uniform(X – 1)	QoS-Profile	X – 1	X	X
	QoS-Layer-Index	X – 1	X	FWD
Pipe(X – 1)	QoS-Profile	X – 1	X	X
	QoS-Layer-Index	X – 1	X	FWD
Short-Pipe(X – 1)	QoS-Profile	X	X	X
	QoS-Layer-Index	X	X	FWD

In the table, the left column shows the previous incoming-interface (X – 1) QoS-Model. The first row shows the current incoming-interface (X) QoS-Model. The match between QoS-Model (X – 1) and QoS-Model (X) determines which QoS-profile and QOS-Layer-Index to use. For example:

- If the previous incoming-interface (X – 1) is Uniform and the current incoming-interface (X) is Pipe, then the QoS-profile and QOS-layer-index is based on the current incoming-interface (X)
- If the previous incoming-interface (X – 1) is Pipe and the current incoming-interface (X) is Uniform, then the QoS-profile and QOS-layer-index is based on the previous incoming-interface (X – 1)

NOTE: Short-Pipe is similar to Pipe but provides the QoS-properties according to the forwarding-layer. This is useful for the example of Egress LSR case, IPoMPLS. Here, the PHB mapping takes the IP DSCP and not the BoS label EXP. Short-Pipe is expected to be used only in BOS.

NOTE: In the L2 MPLS VPN termination scenario, the forwarding layer is Ethernet when labels are terminated. There is always a LIF derived (default LIF if explicitly no hit) for QoS propagation. To select the QoS profile from PWE and the QoS input PCP-DEI from native Ethernet, the QoS profile should be configured as Service Tagged Mode, while the PWE QoS model cannot be configured as short-pipe.

Stuck QoS-Model can be used by the incoming-port such that any new incoming-interface cannot change its QoS properties.

The following example demonstrates how to use QoS-Model:

- In routing (IPoE), a packet with In-AC QoS-Model Pipe and In-RIF QoS-Model Short-Pipe, the QoS-properties are set according to In-RIF.
- In IPv4, IPv6, or MPLS forward, the QoS properties can be set according to In-AC if In-RIF is set to Uniform, as the following table shows.

In-AC QoS Model	**In-RIF QoS Model**	**InLIF1**	**Forward Header**	**Expected QoS Profile**
Pipe	Uniform	XXX	IPv4/IPv6 Forward	In-AC

In-AC QoS Model	In-RIF QoS Model	InLIF1	Forward Header	Expected QoS Profile
Pipe	Uniform	XXX	MPLS Forward	In-AC
Pipe	Uniform	MPLS-LIF1 (Uniform)	IPv4/IPv6 Forward	In-AC

- In bridging (E), a packet with in-AC QoS-Model Pipe and In-Port QoS-Model Short-Pipe, the QoS-properties are set according to In-AC. It is also possible to decide according to In-Port, when In-Port is set to QoS-Model Stuck.
- In L3VPN applications (IPoMPLSoMPLSoE) shown in Figure 3, the cases in Table 3 can be achieved:

Figure 3: QoS-Model in L3VPN Application

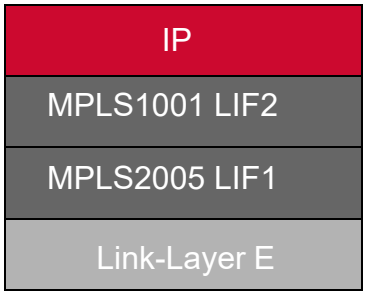


Table 3: QoS-Model L3VPN Cases

LIF1 QoS-Model	LIF2 QoS-Model	Expected QoS-Profile	Expected QOS-Layer
Pipe	Pipe	LIF2	MPLS1001
Uniform	Uniform	LIF1	MPLS2005
Uniform	Pipe	LIF2	MPLS1001
Pipe	Uniform	LIF1	MPLS2005
Short-Pipe	Short-Pipe	LIF2	IP
Uniform	Short-Pipe	LIF2	IP
Short-Pipe	Uniform	LIF2	MPLS1001
Short-Pipe	Pipe	LIF2	MPLS1001
Pipe	Short-Pipe	LIF2	IP

The following table shows the SDK options that exist per incoming-interface:

Object	Supported PHB QoS-Models	Supported Remark QoS-Models
Incoming-Port	Short-Pipe (default), Pipe, Stuck	Short-Pipe (default), Pipe, Stuck
In-AC (`bcm_vlan_port_create`)	Pipe (default)	Pipe (default)
Native In-AC (`bcm_vlan_port_create` with flag NATIVE)	Uniform (default)	Uniform (default)
L3-LIF (MPLS-Tunnel/IP-Tunnel)	Pipe (default), Uniform, Short-Pipe	Pipe, Uniform, Short-Pipe (default)
In-ETH-RIF	Short-Pipe (default), Uniform	Short-Pipe (default), Uniform
PWE	Pipe (default), Uniform, Short-Pipe	Pipe (default), Uniform, Short-Pipe
In-ETH-RIF	Short-Pipe (deafult), Uniform, Pipe	Short-Pipe (default), Uniform, Pipe

While PHB and Remark procedures are two different packet flows, the incoming QoS-Profile is the same input for both procedures. Different objects have different QoS profile sizes.

Table 4: Object QoS Profile Size

Object	QoS Profile Size
In-AC	8 bits that satisfy any of following conditions: ■ P2MP service type with counter statistics, wide data, or protection. ■ P2P service type with counter statistics, wide data, and protection. ■ PON LIF created with the `BCM_VLAN_PORT_FLAGS2_MAC_SAV` flag. Otherwise, the size is 6 bits.
Native In-AC	10 bits when created with the `BCM_VLAN_PORT_VLAN_TRANSLATION` flag. Otherwise, the size is 8 bits.
Control LIF	10 bits
MPLS-Tunnel	8 bits
PPPoE/L2TP tunnel	10 bits
IP-Tunnel	8 bits
PWE/EVPN	10 bits for the P2P service type created with the `BCM_MPLS_PORT2_CROSS_CONNECT` flag. Otherwise, the size is 8 bits.
In-ETH-RIF	4 bits
BIER/SRv6	8 bits

Ingress QoS mapping tables are split into two steps:

1. Given QoS-Profile (PHB/RM-profile) and associated layer types (decided by the packet QoS-Index), select the QoS-Opcode.
2. Given QoS-Opcode and packet-header fields, map it to QoS properties. For PHB, map packet fields to TC and DP. For RM, map packet fields to Nwk-QOS.

In some cases, the QoS-Opcode process is skipped, and the QoS-profile and its associated layer types are mapped directly to the QoS properties without receiving any input from the packet parameters. This procedure is known as *explicit mapping*. Three packet types use explicit mapping:

- L2 untagged packets
- Packets with a QoS profile that comes from the incoming port only
- Unknown packet types with no match to QoS layer information

It is possible to set a few attributes per QOS-profile:

- If the packet is ETH L2 forwarding and its header above is L3 (for example, IP or MPLS), by default, QoS packet parameters are taken from the L2 header. It is possible to set the QOS-profile to ignore L2 packet parameters and to take L3 header parameters instead.
- If the packet is ETH L2 forwarding and has two VLAN tags, it is possible to set which QoS packet parameters should be taken: only the outer-tag (as its default configuration), the inner-tag, or both tags.

In the BCM88670, QoS-profile and QoS-Opcode objects are the same entity. In the BCM88690, a separation exists between QoS-profile and QoS-Opcode objects. QoS-Opcode acts as an intermediate object that allows a better utilization of the mapping between incoming interface QoS requirements and QoS mapping tables. For example, with the BCM88690, it is possible to map two different Incoming-interfaces with the same L2 QoS properties but different L3 QoS properties. To accomplish this, use the same QoS-Opcode ID for the L2 mapping table and use a different QoS-Opcode ID for the L3 mapping table.

During SDK initialization, QoS profile 0 is allocated and used as the default profile for each incoming interface. In order to have a basic default QoS scheme, QoS profile 0 must be configured. See application initialization for reference a default 1:1 mapping per packet type.

NOTE: The current SDK does not support MPLS upstream assignment of QoS.

NOTE: When using the default QoS model, its exact value is not returned. The result is invalid type.

10.3.1 SOC Properties

None

10.3.2 Configuration Flow

1. QoS-profile allocation: Allocate a new QoS-profile by calling: `bcm_qos_map_create(unit, flags, &map_id);`
   - `flags` – `BCM_QOS_MAP_PHB` | `BCM_QOS_MAP_REMARK` | `BCM_QOS_MAP_INGRESS`
   - `flags` can also receive `BCM_QOS_MAP_WITH_ID` – Allow the user to choose the ID allocated, the SDK is adding a prefix to encode the ID.
2. Set QOS-profile attributes: If required, the following attributes can be set per QOS-profile, call:

   `bcm_qos_map_control_set(unit, map_id, flags, type, arg);`
   - `flags` – Either BCM_QOS_MAP_PHB or BCM_QOS_MAP_REMARK.
   - `type`, `arg` – The following table describes the supported `type` and `arg` values.

Type	arg Options	Meaning
`bcmQosMapControlL3L2`	0 – Disable 1 – Enable	Enable or disable the profile is for L2-use-L3. Default is disabled.

`bcmQosMapControlL2TwoTagsMode`	`bcmQosMapControlL2TwoTagsModeDoubleTag`
	`bcmQosMapControlL2TwoTagsModeInnerOnly`
	`bcmQosMapControlL2TwoTagsModeOuterOnly`
		If packet is ETH, QOS parameters derived from outer tag only, inner tag only or double tags. By default is outer tag only.

`bcmQosMapControlMplsPortMode ServiceTag`	`bcmQosMapControlMplsPortModeServiceTagDisable`
	`bcmQosMapControlMplsPortModeServiceTagTypical`
	`bcmQosMapControlMplsPortModeServiceTagOuter`
		If PWE tagged mode, native ETH QOS parameter derived from disabled (take from PWE), outer tag only, or typical for two native tags case.

NOTE:

- For a certain QoS profile, only one control type can work in nondefault mode. In other words, if L2-use-L3 is enabled, `L2TwoTagsMode` must be an outer-tag only, and `MplsPortModeServiceTag` must be disabled.
- If L2-use-L3 is enabled, and the packet L3 is not IPv4, IPv6, or MPLS, QoS input comes from L2 default (outer tag or untagged TC/DP.)
- In the BCM88830, when the MPLS port is in tagged mode and the QoS type is not the expected number of tags in the Native L2 header, EXP is taken as input to the QoS instead of the Native L2 header QoS VLAN tags.

3. QoS-Opcode allocation:
   – Allocate a new QoS-Opcode by calling: `bcm_qos_map_create(unit, flags, &map_id);`
     - `flags` – `qos_app_type | BCM_QOS_MAP_OPCODE | BCM_QOS_MAP_INGRESS`, where `qos_app_type` can be `BCM_QOS_MAP_PHB` *or* `BCM_QOS_MAP_REMARK`
     - `flags` can also receive `BCM_QOS_MAP_WITH_ID` – Allow the user to choose the ID allocated. The SDK adds a prefix to encode the ID.
4. Connect profile and associate layer-types to Opcode by calling*:* `bcm_qos_map_add(unit, flags, &in_map, qos_map_id)`
   – `qos_map_id` – The QoS-Profile ID
   – `in_map.opcode` – QoS-Opcode ID
   – `flags` – `qos_app_type` | `BCM_QOS_MAP_OPCODE` where `qos_app_type` is either `BCM_QOS_MAP_PHB` or `BCM_QOS_MAP_REMARK`
   – The following mapping entries must be set:

Packet Type	additional_flags
L2, inner tag	`BCM_QOS_MAP_L2`, `BCM_QOS_MAP_L2_INNER_TAG`
L2, outer tag	`BCM_QOS_MAP_L2`, `BCM_QOS_MAP_L2_OUTER_TAG`
L2, double tags	`BCM_QOS_MAP_L2`, `BCM_QOS_MAP_L2_TWO_TAGS`
L3, IPv4	`BCM_QOS_MAP_L3, BCM_QOS_MAP_IPV4`
L3, IPv6	`BCM_QOS_MAP_L3, BCM_QOS_MAP_IPV6`
MPLS	`BCM_QOS_MAP_MPLS`
Recycle	`BCM_QOS_MAP_RCH`

5. **Add entries to a specific QoS-opcode** by calling*:* `bcm_qos_map_add(unit, flags, &in_map, qos_map_id)`
   – `qos_map_id` – QoS-Opcode
   – `flags` – `qos_app_type`. Where `qos_app_type` can be `BCM_QOS_MAP_PHB` *or* `BCM_QOS_MAP_REMARK` and `additional_flags` with the map parameters are defined as follows:

Packet Type	additional_flags	Map Parameters	Packet Parameters
L2, inner tag	`BCM_QOS_MAP_L2`, `BCM_QOS_MAP_L2_INNER_TAG`	`in_map.inner_pkt_pri` `in_map.inner_pkt_cfi`	Inner tag PRI, CFI
L2, outer tag	`BCM_QOS_MAP_L2`, `BCM_QOS_MAP_L2_OUTER_TAG`	`in_map.pkt_pri` `in_map.pkt_cfi`	Outer tag PRI, CFI
L2, double tags	`BCM_QOS_MAP_L2`, `BCM_QOS_MAP_L2_TWO_TAGS`	`in_map.pkt_pri` `in_map.pkt_cfi` `in_map.inner_pkt_pri` `in_map.inner_pkt_cfi`	Outer tag PRI and CFI, and inner tag PR I and CFI
L3, IPv4	`BCM_QOS_MAP_L3, BCM_QOS_MAP_IPV4`	`in_map.dscp`	IPv4 TOS
L3, IPv6	`BCM_QOS_MAP_L3, BCM_QOS_MAP_IPV6`	`in_map.dscp`	IPv6 TOS
MPLS	`BCM_QOS_MAP_MPLS`	`in_map.exp`	MPLS exp
RCH PHB	`BCM_QOS_MAP_RCH, BCM_QOS_MAP_PHB`	`in_map.remark_int_pri` (TC: 3b), `in_map.remark_color` (DP: 2b)	TC, DP from RCH header
RCH Remark	`BCM_QOS_MAP_RCH, BCM_QOS_MAP_REMARK`	`in_map.int_pri` (QOS -8b)	QoS from RCH header

- `in_map` result mapping for PHB (TC and DP):
  - `in_map.int_pri` – TC
  - `in_map.color` – DP
- `in_map` result mapping for Remark (NWK-Qos):
  - `in_map.remark_int_pri` – NWK-Qos

6. **Explicit mapping**: To add entries call:`bcm_qos_map_add(unit, flags, &in_map, qos_map_id)`:
   - `flags` – `BCM_QOS_MAP_PHB` | `BCM_QOS_MAP_REMARK` | `BCM_QOS_MAP_INGRESS`
   - `qos_map_id` – QoS-profile

   The following table shows the packet type and the corresponding configuration flags:

Packet Type	additional_flags	Map Params	Packet Params
L2, untagged	`BCM_QOS_MAP_L2,BCM_QOS_MAP_L2_UNTAGGED`	QoS profile	none
Port	`BCM_QOS_MAP_PORT`	QoS profile	none

7. **Connect the Incoming-interface (RIF/VSI/Port/LIF) to the QoS profile**. Possible by:
   - Port/LIF: `bcm_qos_port_map_set(unit, gport, ingress_qos_id, egress_qos_id)`
     - `gport` – Can be local-port or L2/L3 logical-gport (created by LIF APIs)
     - `ingress_qos_id` – QoS-Profile value or -1 (ignore).
     - `egress_qos_id` – QoS-Profile value or -1 (ignore)
   - In-ETH-RIF: by calling: `bcm_l3_ingress_create(unit, l3_ing_if, eth_rif_id)`
     - `l3_ing_if.qos_map_id` = QoS-Profile value.
     - For more information about the API, see Section 22.6, ETH-RIF (L3 VSI Interface) and VRF Definition

8. **QoS-Model Configuration:** Note that QoS-Model is being set by a new structure called `ingress_qos_model`:

```
typedef struct bcm_qos_ingress_model_s {
    bcm_qos_ingress_model_type_t ingress_phb;   /* ingress PHB model */
    bcm_qos_ingress_model_type_t ingress_remark; /* ingress remark model */
    bcm_qos_ingress_model_type_t ingress_ttl;   /* ingress ttl model */
} bcm_qos_model_t;
```

   Each field relates to QoS property (PHB, Remark, and TTL). Each of them can be one of the following:

```
    bcmQosIngressModelInvalid = 0,      /* qos model invalid */
    bcmQosIngressModelShortpipe = 1,    /* qos model is short pipe */
    bcmQosIngressModelPipe = 2,         /* qos model is pipe */
    bcmQosIngressModelUniform = 3,      /* qos model is uniform */
```

   Setting `bcmQosIngressModelInvalid` to all fields will set the default configuration.

   Also upon `get`, if the default configuration is set, `bcmQosIngressModelInvalid` will be set.

   The following options are possible to configure.

Object	API	QoS-Model
Incoming-Port	N/A	Call `bcm_port_control_set()` with `type=` `bcmPortControlIngressQosModelPhb` `bcmPortControlIngressQosModelRemark`
L2 interface	N/A	N/A
L3 interface	`bcm_l3_intf_create`	Call with `ingress_qos_model` in `bcm_l3_intf_t`
MPLS-Tunnel	`bcm_mpls_tunnel_switch_create`	Call with `ingress_qos_model` in `bcm_mpls_tunnel_switch_t`
PWE	`bcm_mpls_port_add`	Call with `ingress_qos_model` in `bcm_mpls_port_t`
IP-Tunnel	`bcm_tunnel_terminator_create`	Call with `ingress_qos_model` in `bcm_tunnel_terminator_t`

The following table shows the default QoS model for LIFs.

Object	PHB	Remark	TTL
L2 interface	Pipe	Pipe	Pipe
L2 native interface	Uniform	Uniform	Uniform
L3 interface	Short Pipe	Short Pipe	Short Pipe
MPLS-Tunnel	Pipe	Short Pipe	Short Pipe
PWE	Pipe	Pipe	Pipe
IP-Tunnel	Pipe	Short Pipe	Short Pipe

10.3.3 Shell Commands

None

10.3.4 Application Reference

QoS Default Initialization Values

Example of application that a default 1:1 mapping for the QoS values for all QoS applications (PHB, remark, ECN) for each packet type

- **Type:** Default application reference
- **Path:**
  `$SDK/src/appl/reference/dnx/appl_ref_qos_init.c`
  Function `appl_dnx_qos_init()`

Example of PHB/Remarking

- **Type**: CINT reference
- **Path**:
  `$SDK/src/examples/dnx/cint_dnx_qos_l2_phb.c`
  `$SDK/src/examples/dnx/cint_dnx_qos_l3_remark.c`
  `$SDK/src/examples/dnx/cint_dnx_mpls_qos.c`

10.4 Ingress VLAN Translation VLAN-PCP QoS

As part of the Ingress VLAN translation mechanism (IVE), it is possible to edit PCP, DEI of the Ethernet VLAN headers. QoS mapping of Ingress VLAN translation is done at the ingress device according to Incoming-AC QoS profile, and its results are passed to the egress device. At the egress device, according to the IVE decision, the resulting PCP, DEI values are taken or not (and in which order, if there are two tags). The QoS-profile for VLAN-translation is also known as *VLAN-PCP-QOS-profile*.

For more information about Ingress VLAN translation, see Chapter 16, VLAN Editing Mechanism.

10.4.1 SOC Properties

None

10.4.2 Configuration Flow

1. Ingress VLAN-PCP-QOS-Profile:
   – Allocate a new VLAN-PCP-QOS-profile by calling: `bcm_qos_map_create(unit, flags, &map_id);`
      - `flags` – `BCM_QOS_MAP_L2_VLAN_PCP` | `BCM_QOS_MAP_INGRESS`
      - `flags` can also receive `BCM_QOS_MAP_WITH_ID` – Allow the user to choose the ID allocated, the SDK is adding suffix to encode the ID.
2. **Ingress VLAN-PCP-QOS-Mapping:**
   – Add entries to a specific VLAN-PCP-QOS-profile by calling*:* `bcm_qos_map_add(unit, flags, &map, qos_map_id)`
      - `qos_map_id` – The created VLAN-PCP-QOS-profile
      - `flags` – `BCM_QOS_MAP_L2_VLAN_PCP, BCM_QOS_MAP_INGRESS` and `additional_flags` with the map parameters define as follows:

Packet type	additional_flags	In-Map Parameters	Out-Map Parameters
L2 tagged	`BCM_QOS_MAP_L2`	`map.color` (DEI) `map.int_pri` (PCP)	`map.pkt_cfi` (DEI) `map.pkt_pri` (PCP)
L2 untagged	`BCM_QOS_MAP_L2_UNTAGGED`	`map.int_pri` (TC) `map.color` (DP)	`map.pkt_cfi` (DEI) `map.pkt_pri` (PCP) `map.inner_pkt_cfi` (Inner DEI) `map.inner_pkt_pri` (Inner PCP)

NOTE: For the resulting PCP-DEI values, the decision is the same for the L2-tagged outer and inner tags.

3. Connect Incoming-interface (Incoming-AC) to Ingress VLAN-PCP-QOS-profile. Possible by:
   – `bcm_qos_port_map_set(unit, gport, ingress_qos_id, -1)`
      - `gport` – The created VLAN-Port (vlan_port_id from bcm_vlan_port_create)
      - `ingress_qos_id` – VLAN-PCP-QOS-Profile value.

10.4.3 Shell Commands

None

10.4.4 Application Reference

Example of ingress VLAN translation PCP mapping

- Type: CINT reference
- Path: `$SDK/src/examples/dnx/cint_qos_l2_vlan_edit.c`

10.5 Ingress VLAN Translation Policer QoS

As part of the ingress VLAN translation mechanism (IVE), it is possible to change the final decision of PCP, DEI according to resulted PCP, DEI, FTMH Drop-Precedence (DP) and the Incoming-interface (InLIF) QoS profile. This QoS profile is also known as *Ingress-Policer-QOS-profile*.

This functionality is useful in order to influence PCP, DEI values according to Policer decision that is passed in the system header (FTMH DP)

The mapping is done the same for both Outer-Tag and Inner-Tag. The incoming interface can provide two different Ingress-Policer-QOS-profiles: one for Outer-Tag and one for Inner-Tag. According to Ingress-Policer-QOS-profile, the PCP-DEI and FTMH-DP are mapped to a new PCP-DEI that is used for the IVE action.

It is possible to have up to four different Ingress-Policer-QOS-profiles. Profile 0 is allocated at SDK initialization time to indicate transparent mapping between the PCP-DEI results to the new PCP-DEI after IVE action (with no DP effect).

NOTE: While the functionality is described as ingress VLAN translation, the actual configuration is split between the ingress and egress devices. The ingress device is responsible for mapping Incoming-interface to Ingress-Policer-QOS-profile. The egress device is responsible for the QoS mapping tables.

10.5.1 SOC Properties

To allocate more than one value of Ingress-Policer-QOS-profile, allocating resources from InLIF-profile is required:

- `in_lif_profile_egress_allocate_policer_inner_dp` – Indicate how many Inner-Tag profiles need to be supported. By default 1.
- `in_lif_profile_egress_allocate_policer_outer_dp` – Indicate how many Outer-Tag profiles need to be supported. By default 1.

For more information about LIF-profile resources and management, see Chapter 7, Logical Interface (LIF) Management.

10.5.2 Configuration Flow

1. At both the ingress and egress device, Ingress Policer-QOS-Profile:
   - Allocate a new Ingress Policer-QOS-profile by calling: `bcm_qos_map_create(unit, flags, &map_id);`
     - `flags` – `BCM_QOS_MAP_POLICER | BCM_QOS_MAP_INGRESS`
     - `flags` can also receive `BCM_QOS_MAP_WITH_ID` – Allow the user to choose the ID allocated, the SDK is adding suffix to encode the ID.

   By default profile 0 is allocated and used as the default value on Incoming-LIF.
2. At the egress device, Ingress Policer-QOS-Mapping:
   - Add entries to a specific Policer-QOS-profile by calling: `bcm_qos_map_add(unit, flags, &map, qos_map_id)`
     - `flags` – `BCM_QOS_MAP_POLICER`, `BCM_QOS_MAP_INGRESS`
     - `map.color` – (In) FTMH DP
     - `map.remark_int_pri` – (In) PCP-DEI (4bits)
     - `map.pkt_pri` – (Out) New PCP
     - `map.pkt_cfi` – (Out) New DEI
     - `qos_map_id` – The created Policer-QOS-profile

3. At the ingress device, connect Incoming-interface (Incoming-LIF) to Ingress Policer-QOS-profile. Possible by:
   – Connect to Outer-Tag profile by `bcm_port_control_set(unit, gport, bcmPortControlOuterPolicerRemark, qos_profile)`
     - `gport` – The Incoming-interface (LIF), created by LIF APIs (for example, `vlan_port_id`, `mpls_port_id`)
     - `qos_profile` – Ingress Policer-QOS-profile
   – Connect to Inner-Tag profile by `bcm_port_control_set(unit, gport, bcmPortControlInnerPolicerRemark, qos_profile)`
     - `gport` – The Incoming-interface (LIF), created by LIF APIs (for example, `vlan_port_id`, `mpls_port_id`)
     - `qos_profile` – Ingress Policer-QOS-profile

10.5.3 Shell Commands

None

10.5.4 Application Reference

Example of ingress policer QoS:

- Type: CINT reference
- Path: `$SDK/src/examples/dnx/qos/cint_dnx_qos_ingress_policer.c`

10.6 Explicit Congestion Notification

Explicit Congestion Notification (ECN) is an extension to the IP, TCP, and other protocols, that allows end-to-end congestion notification without dropping packets as part of the protocol. ECN can be supported if it was negotiated successfully by the two endpoints. Two bits of the DiffServ field of IPv4/IPv6 stores ECN capability and congestion indication. The ingress ECN section includes the classic ECN and L4S (Low Latency, Low Losses) features.

At Ingress, the user can set the ingress QoS model to indicate if ECN is eligible on the incoming interface (for example, IP-Tunnel, In-RIF). If the service is an L2 bridge and the upper-layer protocol is IP, set ECN eligibility by using ACLs.

Following IP.ECN encoding, the SDK is as follows:

- Packet IP.ECN 2b00 means it is not ECN capable.
- Packet IP.ECN 2b10 means it is classified as a classic ECN.
- Packet IP.ECN 2b01 means it is classified to L4S.
- Packet IP.ECN 2b11 means congestion.

If ECN is eligible and the packet is ECN capable, CNI may be updated if traffic triggers the threshold. For threshold configurations, see the *Traffic Manager Programming Guide*.

NOTE:

- MPLS ECN is not supported.
- BCM88690 only: If the packet IP.ECN is 2b01, then the packet is classified to classic ECN.

At the egress, the ECN information is updated with the CNI and ECN capability. The ECN can be remarked at the original locations and, possibly, marked in DSCP in IP tunnels. Currently, only the following scenarios support marking/remarking ECN:

- L2 bridge and the upper-layer is IP
- IPv4 and IPv6 routing
- IPv4 and IPv6 VXLAN tunnel

For an L2 bridge where the upper-layer is IP, `egress_ecn` of `egress_qos_model` indicates whether the layer plus one ECN eligible is TRUE or FALSE. For an IP routing service, if ECN is eligible for in-RIF, the egress forward layer is ECN eligible as well. For IPv4 and IPv6 VXLAN tunnel, `egress_ecn` of `egress_qos_model` indicates whether ECN eligible is TRUE or FALSE. If a layer is ECN eligible, egress marking/remark should be configured with ECN. In the egress marking/remark mapping, only 6 MSBs for DSCP mapping are valid. The 2-LSB ECN is marked with device packet ECN eligible and the status of CNI originated from the system headers.

NOTE: For ECN marking and remarking, map the remark-profile with dedicated ECN opcodes to make it functional. By default, the remark-profile to ECN is assigned to opcodes 0 to 3, which are not initialized correctly. It is expected to configure them correctly if ECN is used in the system. For an example, refer to the application reference (see Section 10.6.4, Application Reference).

10.6.1 SOC Properties

None

10.6.2 Configuration Flow

Ingress ECN

To enable ECN on an incoming interface, the ECN model must be set as eligible:

```
object.ingress_qos_model.ingress_ecn = bcmQosIngressEcnModelEligible.
```

The objects in the following table can be configured.

Object	API	ECN Model
Incoming-Port	N/A	N/A
L2 interface	N/A	N/A
L3 interface	`bcm_l3_intf_create`	`intf.ingress_qos_model.ingress_ecn`
MPLS-Tunnel	N/A	N/A
PWE	N/A	N/A
IP-Tunnel	`bcm_tunnel_terminator_create`	`info.ingress_qos_model.ingress_ecn`

Egress ECN

1. For remark profile allocation and mapping entry, see Section 10.10, Egress Marking, Remarking, and Inheritance.

```
ecn_eligible in egress_qos_model indicates that layer ECN eligible is TRUE or FALSE.
  typedef struct bcm_qos_egress_model_s {
    bcm_qos_egress_model_type_t egress_qos; /* egress qos variable */
    bcm_qos_egress_model_type_t egress_ttl; /* egress ttl */
    uint8 ecn_eligible;                     /* layer ecn is eligible */
} bcm_qos_egress_model_t
```

2. To set egress layer (IP tunnel) ECN as eligible, the egress QoS model should be set. The default ECN is `bcmQosEgressEcnModelInvalid`.

```
typedef enum bcm_qos_egress_ecn_model_type_e{
    bcmQosEgressEcnModelInvalid = 0, /* qos ecn model is invalid */
    bcmQosEgressEcnModelEligible = 1 /* qos ecn model is eligible */
 }bcm_qos_egress_ecn_model_type_t;
```

10.6.3 Shell Commands

None

10.6.4 Application Reference

Example for ingress ECN enable on RIF:

- **Description**: Example of application to enable ECN on InRIF
- **Type**: CINT reference
- **Path**: `$SDK/src/examples/dnx/qos/cint_dnx_qos_ecn.c`
- **Function**: `dnx_ecn_ingress_eligible_set()`

Example for ingress ECN enable on IP tunnel:

- **Description**: Example of application to enable ECN on termination IP tunnel
- **Type:** CINT reference
- **Path:** `$SDK/src/examples/sand/cint_ip_tunnel_term_basic.c`
- **Function**: `ip_tunnel_term_update_ecn_eligible()`

Example for IP routing egress ECN:

- **Description**: Example of IP routing application egress ECN
- **Type**: CINT reference
- **Path**: `$SDK/src/examples/dnx/qos/cint_dnx_qos_ecn.c`
- **Function**: `dnx_ecn_basic_example()`

Example for egress ECN enable on IP VXLAN GPE tunnel:

- **Description:** Example of application to enable ECN on termination IP tunnel
- **Type:** CINT reference
- **Path:** `$SDK/src/examples/sand/cint_ip_tunnel_encap_basic.c`
- **Function**: `ip_tunnel_encap_basic_qos_update()`

10.7 Ingress TTL

Ingress-TTL processing follows a similar method as Ingress PHB/Remarking. Given the QoS-Model (Pipe, Uniform, and Short-Pipe), the Ingress-TTL value is decided according to the QOS-Layer-Index. If there is no TTL field in the layer header (for example, in the Ethernet header), by default, the maximum TTL value of 255 is used and passed to the egress by the internal system header.

10.7.1 SoC Properties

None

10.7.2 Configuration Flow

TTL-Model Configuration: See the Ingress PHB/Remarking configuration (Section 10.3.2, Configuration Flow).

10.7.3 Shell Commands

None

10.7.4 Application Reference

Ingress-TTL Application Examples

Example of ingress TTL model is uniform.

- **Type**: CINT reference
- **Path**:
  `$SDK/src/examples/sand/cint_ip_route_basic_qos.c`
  `$SDK/src/examples/sand/cint_vpls_roo_basic.c`

10.8 Egress Policer QoS

It is possible to map QoS meter to offset to the base meter pointer. For more information, see Section 15.3, Egress QoS Meter Offset.

10.9 Egress OAM QoS

It is possible to map OAM PCP offset to the base counter pointer by QoS mapping. For more information see Chapter 33, Operations, Administration, and Maintenance.

10.10 Egress Marking, Remarking, and Inheritance

Egress-QoS processing is split into two parts:

- Marking and remarking – Determine the QoS of existing network header editing (remarking) and newly added network headers (marking)
- Inheritance – Determine the next-layer QoS

In all cases, the processing of Egress-QoS starts with the initial NWK-QOS as the first input. For BCM88690 system mode, the initial NWK-QOS comes directly from the system header (i.e. from the Ingress NWK-QOS remarking decision and FTMH-DP).

As part of the egress processing, the following parameters influence the Marking/Remarking/Inheritance of the headers.

- QoS-Model – The following options exist:
  - Pipe-Next-Namespace: Set a new NWK-QOS from the current layer and translate it with the subsequent encapsulate entry.
  - Pipe-My-Namespace: Set a new NWK-QOS from the current layer without translation. The QoS-Model influences all procedures (Marking/Remarking/Inheritance).
  - Uniform: Use previous NWK-QOS and translate it with the subsequent encapsulate entry.
  - Initial: Use NWK-QOS in the system header and translate it with the subsequent encapsulate entry.
- QoS-Model influences all procedures.
- NWK-QOS parameters – For a Pipe model, determine the new NWK-QOS according to the parameters given in the current layer.
- Remark-Profile (QoS-profile) – Taken from the subsequent encapsulate entry to select a new translated NWK-QOS according to the outgoing interface.

  During initialization, QOS-profile 0 is allocated and it is used as the default profile on LIF creation. For an example configuration of QOS-profile 0, see Section 10.10.6, Application Reference.

All the above parameters are derived according to the OutLIF/Out-ETH-RIF entry.

According to QoS-Model, the NWK-QOS before mapping is decided. It can be a new NWK-QOS that is set according to NWK-properties of the LIF entry only (pipe) or from previous NWK-QOS (uniform).

Then two different mappings apply :

- Inheritance: NWK-QOS before mapping is used together with Remark-profile to translate the NWK-QOS to the next encap stage (next outgoing interface).
- Marking/Remarking: Remark-profile and packet-layer-type provide an intermediate object QoS Opcode. Then NWK-QoS before mapping is used together with QoS opcode to decide the QoS-VAR, which is set to the QoS network headers.

  Note that with remarking, for routing packets (IPv4/IPv6), it is possible to preserve the original QoS field in the packet header. This is done by setting a special *preserve* opcode.

10.10.1 Forwarding Plus One Remarking

Generally, the remarking/marking procedure will update the QoS value in the forwarding header and newly encapsulated headers. It is also possible to remark the subsequent IP header with a new QoS value (DSCP for IPv4 headers, TC for IPv6 headers) when the forwarding header is Ethernet. This is called *forwarding plus one remarking*. The new QoS value is also mapped from NWK-QOS but with a different set of QoS mapping configurations than forwarding and encapsulated headers.

By default, forwarding plus one remarking is disabled in all cases. It is possible to enable it by mapping the object remark-profile (in that case, it is usually VLAN-Port) with the flag `BCM_QOS_MAP_L3_L2` to a valid QoS opcode and not the preserve opcode (`BCM_QOS_OPCODE_PRESERVE`) with flags. By default, if no mapping is done, opcode preserve is used.

10.10.2 Routing DSCP Preserve

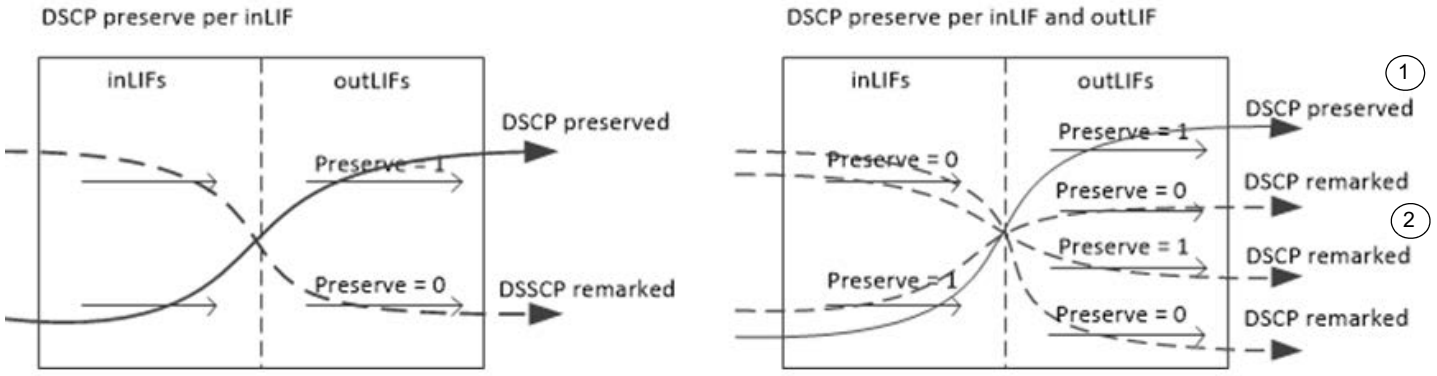
In some cases, it is possible to preserve the DSCP/TC field in the forwarding IP header, while the mapped network QoS is still required in the latter tunnel encapsulation.

The system can work in one of the two DSCP preserve modes, which is selected by an SOC property:

- Per OutLIF (default mode)
- Per InLIF and OutLIF

If the system is working in DSCP preserve per InLIF and OutLIF mode, to achieve DSCP preserve, the enabler should be set in both the InLIF and OutLIF. For any other case, DSCP will be remarked.

Figure 4: DSCP Preserve



Regarding the configuration of DSCP preserve enabler per OutLIF:

- This is done by mapping the OutLIF's remark profile to an QoS opcode reserve value.
- If the DSCP preserve mode is *per InLIF and OutLIF*, the same OutLIF may be used for both DSCP preserve (case 1 in the figure) and remarking (case 2 in the figure). In this case, the remark-profile should be mapped to a valid opcode for QoS remarking first, and the QoS opcode reserve value is for DSCP preserve.

Regarding the configuration of DSCP preserve enabler to InLIF:

- If the system is working in BCM88690 system-headers mode, the ingress field processor is required to enable the feature on the correct InLIF. The user can use any available indicators from the InLIF for the ingress field processor to recognize it, InLIF-profile, wide-data, and so on. This section uses the InLIF profile as an example to introduce the configuration flow.

10.10.3 SOC Properties

`logical_port_routing_preserve_dscp` – DSCP preserve modes are available.

Modes	SOC Property	Enabler Carrier
Per OutLIF	`logical_port_routing_preserve_dscp=2`	`outLIF.remark_profile`
Per InLIF and OutLIF	`logical_port_routing_preserve_dscp=1`	`inLIF.inlif_profile` `outLIF.remark_profile`

10.10.4 Configuration Flow

1. Allocate QoS-profile (Remark-Profile) for Marking/Remarking/Inheritance: Allocate a new QoS-profile by calling: `bcm_qos_map_create(unit, flags, &map_id);`
   - `BCM_QOS_MAP_REMARK` | `BCM_QOS_MAP_EGRESS`
   - `flags` can also receive `BCM_QOS_MAP_WITH_ID` – Allow the user to choose the ID allocated, the SDK is adding suffix to encode the ID.
2. Allocate QoS-opcode for Marking/Remarking only:
   - Allocate a new QoS-Opcode by calling: `bcm_qos_map_create(unit, flags, &map_id);`
     - `flags`—`BCM_QOS_MAP_REMARK` | `BCM_QOS_MAP_OPCODE` | `BCM_QOS_MAP_EGRESS`
     - `flags` can also receive `BCM_QOS_MAP_WITH_ID` – Allow the user to choose the ID allocated, the SDK is adding suffix to encode the ID.
     - ECN: If ECN QOS-Opcode is required for objects (such as IPv4 and IPv6 tunnels) with ECN eligible, add the flag: `BCM_QOS_MAP_ECN`. When flag is used, four consecutive opcodes are allocated from the general opcode pool for ECN mapping, the API returns the base opcode only.
3. Associate Remark-profile and Packet-Type to a specific opcode for Marking/Remarking only by calling:
   `bcm_qos_map_add(unit, flags, &in_map, qos_map_id)`
   - `qos_map_id` – Remark-profile
   - `in_map.opcode` – QoS-Opcode

     Note that for L3 routing packet types (IPv4, IPv6), it is possible to preserve QoS field in packet header by setting opcode to `BCM_QOS_OPCODE_PRESERVE`. In this case, no need for creating opcode and adding entries per QoS-opcode.

– `flags` – `BCM_QOS_MAP_OPCODE | BCM_QOS_MAP_REMARK` and `additional_flags` with the map parameters defined as follows:

Packet Type	Additional Flags	Description
L2	`BCM_QOS_MAP_L2`	Remarking Ethernet header.
L2 FWD, FWD + 1 IPv4	`BCM_QOS_MAP_L3_L2,BCM_QOS_MAP_IPV4`	Remarking IPv4 TOS in bridging.
L2 FWD, FWD + 1 IPv6	`BCM_QOS_MAP_L3_L2,BCM_QOS_MAP_IPV6`	Remarking IPv6 TC in bridging.
L3, IPv4	`BCM_QOS_MAP_L3, BCM_QOS_MAP_IPV4`	Remarking IPv4 TOS.
L3, IPv6	`BCM_QOS_MAP_L3, BCM_QOS_MAP_IPV6`	Remarking IPv6 TC.
MPLS	`BCM_QOS_MAP_MPLS`	Remarking MPLS EXP.
L3, IPv4 with ECN	`BCM_QOS_MAP_L3, BCM_QOS_MAP_IPV4,BCM_QOS_MAP_ECN`	Remarking IPv4 with ECN eligible.
L3, IPv6 with ECN	`BCM_QOS_MAP_L3, BCM_QOS_MAP_IPV6,BCM_QOS_MAP_ECN`	Remarking IPv6 with ECN eligible.

4. Add entries for Inheritance per QoS-profile.

   Add entries by calling `bcm_qos_map_add(unit, flags, &in_map, qos_map_id)`
   – `qos_map_id` – Remark-profile (used for Remarking)
   – `in_map` key is:
     - `in_map.int_pri` – The NWK-QOS
     - `in_map.color` – DP (Color)
   – `in_map` result:
     - `in_map.remark_int_pri` – The remarking new NWK-QOS (for next layer)
5. Add entries for Remarking/Marking per QoS-opcode: Add entries by calling `bcm_qos_map_add(unit, flags, &in_map, qos_map_id)`
   – `qos_map_id` – QoS-Opcode
   – `in_map` key is:
     - `in_map.int_pri` – The NWK-QOS
     - `in_map.color` – DP (Color)
   – `in_map` result:
     - Packet-header parameters according to the flags:

Packet Type	Additional Flags	Map Result Params	Packet Params
L2, Inner-tag	`BCM_QOS_MAP_L2` `BCM_QOS_MAP_L2_INNER_TAG`	`in_map.inner_pkt_pri` `in_map.inner_pkt_cfi`	PRI, CFI of inner tag
L2, Outer-tag	`BCM_QOS_MAP_L2` `BCM_QOS_MAP_L2_OUTER_TAG`	`in_map.pkt_pri` `in_map.pkt_cfi`	PRI, CFI of outer tag

L2, double-tags	`BCM_QOS_MAP_L2`
	`BCM_QOS_MAP_L2_TWO_TAGS`
		`in_map.pkt_pri`
		`in_map.pkt_cfi`
		`in_map.inner_pkt_pri`
		`in_map.inner_pkt_cfi`
			PRI, CFI of both outer and inner tag

L3, IPv4	`BCM_QOS_MAP_L3, BCM_QOS_MAP_IPV4`	`in_map.dscp`	IPv4 TOS
L3, IPv4, ECN	`BCM_QOS_MAP_L3, BCM_QOS_MAP_IPV4,` `BCM_QOS_MAP_ECN`	`in_map.dscp`	IPv4 TOS[a]
L3, IPv6	`BCM_QOS_MAP_L3, BCM_QOS_MAP_IPV6`	`in_map.dscp`	IPv6 TC
L3, IPv6, ECN	`BCM_QOS_MAP_L3, BCM_QOS_MAP_IPV6,` `BCM_QOS_MAP_ECN`	`in_map.dscp`	IPv6 TC[a]

Packet Type	Additional Flags	Map Result Params	Packet Params
MPLS	`BCM_QOS_MAP_MPLS`	`in_map.exp`	MPLS exp
MPLS	`BCM_QOS_MAP_MPLS_SECOND` **NOTE:** When encapsulating two labels per EEDB entry, `BCM_QOS_MAP_MPLS` is only used to set `Label_0.exp` mapping, `BCM_QOS_MAP_MPLS_SECOND` flag is used to set `Label_1.exp` mapping.	`in_map.exp`	MPLS exp

a. If ECN mapping is used, IPv4 TOS/IPv6 TC is not only mapped from In_dscp and color, but also mapped from ECN indications as shown in the following table (aligned to ECN standard).

FTMH.CNI	FTMH.ECN-Eligible	IPv4.QOS/IPv6.TC 2 lsbs
1	1	3
X	0	0
0	1	1

6. Connect Outgoing-interface (RIF/LIF) to QoS-profile. Possible by:
   - LIF: `bcm_qos_port_map_set(unit, gport, ingress_qos_id, egress_qos_id)`
     - `gport` – Can be L2/L3 logical-gport (created by LIF APIs)
     - `ingress_qos_id` – The value is –1 (ignore)
     - `egress_qos_id` – QoS-Profile value
     - It might be possible to connect LIF to QoS-profile also on the object creation API. See more information on the object creation sequence.
   - MPLS Ingress SWAP operation: See Section 10.15, MPLS Ingress Swap QoS.
   - Out-ETH-RIF: by calling:`bcm_l3_intf_create(unit, intf)`
     - `intf.l3a_intf_id` – ETH-RIF ID
     - `intf.dscp_qos.qos_map_id` – Remark-profile
     - For more information about the API see Section 22.6, ETH-RIF (L3 VSI Interface) and VRF Definition
7. QoS-Model Configuration: The options in the following table are possible to configure:
   - `bcmQosEgressModel` – QOS-Model
   - `bcmQosEgressModelPipeNextNameSpace` – Pipe-Next-Namespace
   - `bcmQosEgressModelPipeMyNameSpace` – Pipe-My-Namespace
   - `bcmQosEgressModelUnfirom` – Uniform

Object	API	BCM QoS-Model Enum
L3 interface	`bcm_l3_intf_create`	Always uniform
MPLS-Tunnel	`bcm_mpls_tunnel_initiator_create`	`label_array[0].egress_qos_model.egress_qos` **NOTE:** For more information regarding MPLS PHP, see Section 10.11, Egress MPLS PHP QoS.
PWE	`bcm_mpls_port_add`	`egress_label.egress_qos_model.egress_qos`
IP-Tunnel	`bcm_tunnel_initiator_create`	`tunnel.egress_qos_model.egress_qos` ■
Egress AC	`bcm_vlan_port_create`	`vlan_port.egress_qos_model.egress_qos`
ARP	`bcm_l3_egress_create`	`egr.egress_qos_model.egress_qos`

8. NWK-QOS Pipe properties: If the QOS-Model is Pipe, the parameters in the following table determine the new NWK-QoS before mapping.

Object	API	New NWK-QOS Before Mapping
MPLS-Tunnel	`bcm_mpls_tunnel_initiator_create`	`label_array[0].exp`
PWE	`bcm_mpls_port_add`	`egress_label.exp`
IP-Tunnel	`bcm_tunnel_initiator_create`	`tunnel.dscp` ■
Egress AC	`bcm_vlan_port_create`	`vlan_port.pkt_pri` and `vlan_port.pkt_cfi`
ARP	`bcm_l3_egress_create`	`egr.mpls_pkt_pri` and `egr.mpls_pkt_cfi`

9. Enable DSCP preserve in the InLIF. If the system is working in BCM88690 system-headers mode, the ingress field processor is used.
   – For BCM88690 system-headers mode, it is required to enable the feature in Forwarding-layer-Additional-Information (FAI) per InLIF.

     The following Field-Processors exist:

     Set a special indicator to the InLIF-profile to this type of InLIF by calling `bcm_port_class_set` `pclass` `bcmPortClassFieldIngressVport`. This sets the value to be used as a marker for iPMF recognition. Call Ingress Field Processor to update FAI. For example:

     Preselector

     Qualifier 1: `bcmFieldQualifyInVportClass`.

     Qualifier 2: `bcmFieldQualifyForwardingType`.

     Field group type: `bcmFieldGroupTypeDirectExtraction`.

     Field group qualifier:

     Qualifier 1: UDF qualifier with FAI.

     Qualifier 2: Constant value of 1.

     Field group action – `bcmFieldActionForwardingAdditionalInfo`.

     The complete example can be found in the references listed in the application reference section.

     For more details on the usage of the field processor, see Chapter 11, Field Processor.

10. For DSCP preserve per OutLIF, see Step 3.

10.10.5 Shell Commands

None

10.10.6 Application Reference

QoS Default Initialization Values

Example of application that a default 1:1 mapping for the QoS values for all QoS applications (PHB, remark, ECN) for each packet type

- **Type**: Default Application Reference
- **Path**: `$SDK/src/appl/reference/dnx/appl_ref_qos_init.c`
  Function `appl_dnx_qos_init()`

Example of PHB L2 usage

- **Type:** CINT reference
- **Path:** `$SDK/src/examples/sand/cint_qos_l2_phb.c`

Example of L3 usage

- **Type:** CINT reference
- **Path:** `cint_qos_l3_remark.c`

Example of MPLS

- **Type:** CINT reference
- **Path:** `$SDK/src/examples/sand/cint_mpls_qos.c`

Example of `PipeMyNameSpace` QoS model

- **Type**: CINT reference
- **Path**: `$SDK/src/examples/dnx/vpls/cint_vpls_mpls_qos.c`

Example of QoS preserving for L3 usage

- **Type:** CINT reference
- **Path:** `$SDK/src/examples/sand/cint_ip_route_tunnel.c`

Example of QoS ECN

- **Type:** CINT reference
- **Path:** `$SDK/src/examples/dnx/qos/cint_dnx_qos_ecn.c`

Example of IP header remarking with bridging.

- **Type:** CINT reference
- **Path:** `$SDK/src/examples/sand/cint_l2_basic_bridge_with_vlan_editing.c`

```
qos_map_l2_fwd_plus_one_remark(int unit, int egress_remark_profile, int opcode, int is_ipv6);
```

Example of DSCP preserve per OutLIF

- **Type:** CINT reference
- **Path:** `$SDK/src/examples/sand/cint_qos_l3_remark.c`

```
qos_l3_dscp_preserve_per_outlif_config(int unit, int egr_port_or_qos_profile, int enable)
```

Example of DSCP preserve per InLIF and OutLIF

- **Type:** CINT reference
- **Path:** `$SDK/src/examples/sand/cint_qos_l3_remark.c`

```
qos_l3_dscp_preserve_per_inlif_outlif_config(int unit, int ing_port, int
egr_port_or_qos_profile)
```

10.11 Egress MPLS PHP QoS

Egress MPLS PHP supports two models for QoS and TTL:

- Uniform
- Pipe

If the QoS model is uniform, NWK-QOS is the egress initial nwk-qos.

If the QoS model is pipe, there are up to eight profiles for QoS mapping. Among the profiles, profile 0 is reserved as 1:1 mapping.

If the PHP TTL model is uniform, TTL is the PHP label TTL.

If the PHP TTL model is pipe, TTL is from the upper layer header.

API (Action PHP)	QoS-Model	TTL-Model
`bcm_mpls_tunnel_switch_create`	info → `egress_label.egress_qos_model.egress_qos = bcmQosEgressModelPipeNextNameSpace` Set to PIPE, default Uniform	info → `egress_label.egress_qos_model.egress_ttl = bcmQosEgressModelPipeMyNameSpace` Set to PIPE, default Uniform
`bcm_mpls_tunnel_initiator_create`	`label_array[0].egress_qos_model.egress_qos = bcmQosEgressModelPipeNextNameSpace` Set to PIPE, default Uniform	`label_array[0].egress_qos_model.egress_ttl = bcmQosEgressModelPipeMyNameSpace` Set to PIPE, default Uniform
`bcm_l3_egress_create`	`egr.egress_qos_model.egress_qos = bcmQosEgressModelPipeNextNameSpace` Set to PIPE, else set to Uniform;	`egr.egress_label.egress_qos_model.egress_ttl = bcmQosEgressModelPipeMyNameSpace` Set to PIPE, else set to uniform

10.11.1 SOC Properties

None

10.11.2 Configuration Flow

1. Allocate QoS-profile (MPLS-PHP-profile): Allocate a new QoS-profile by calling: `bcm_qos_map_create(unit, flags, &map_id);`
   - `BCM_QOS_MAP_MPLS_PHP` | `BCM_QOS_MAP_EGRESS`
   - `flags` can also receive `BCM_QOS_MAP_WITH_ID` – Allow the user to choose the ID allocated, the SDK is adding suffix to encode the ID.
2. Allocate and Add entries with QoS-opcode for Marking only:
   - Allocate a new QoS-Opcode by calling: `bcm_qos_map_create(unit, flags, &map_id);`
     - `flags` = `BCM_QOS_MAP_MPLS_PHP` | `BCM_QOS_MAP_OPCODE` | `BCM_QOS_MAP_EGRESS`
     - `flags` can also receive `BCM_QOS_MAP_WITH_ID` – Allow the user to choose the ID allocated, the SDK is adding suffix to encode the ID.
   - Connect MPLS-PHP-profile and Packet-Type to a specific opcode by calling: `bcm_qos_map_add(unit, flags, &in_map, qos_map_id)`
     - `qos_map_id` – MPLS-PHP-profile
     - `in_map.opcode` – QoS-Opcode

- `flags` – `BCM_QOS_MAP_OPCODE` | `BCM_QOS_MAP_MPLS_PHP` and `additional_flags` with the map params defined in the following table.

Packet Type	Additional Flags
L2	`BCM_QOS_MAP_L2`
L3, IPv4	`BCM_QOS_MAP_L3, BCM_QOS_MAP_IPV4`
L3, IPv6	`BCM_QOS_MAP_L3, BCM_QOS_MAP_IPV6`
MPLS	`BCM_QOS_MAP_MPLS`

3. Add entries by calling `bcm_qos_map_add(unit, flags, &in_map, qos_map_id)`

10.11.3 Shell Commands

None

10.11.4 Application Reference

Qos default initialization values

Example of application that a default 1:1 mapping for the QoS values for PHP

- **Type:** Default Application Reference
- **Path:**
  `$SDK/src/appl/reference/dnx/appl_ref_qos_init.c`
  Function `appl_dnx_qos_init()`

Example of MPLS PHP usage

- **Type:** CINT reference
- **Path:** `$SDK/src/examples/dnx/qos/cint_dnx_qos_php.c`

10.12 Egress TTL

Egress-TTL processing allows either keeping the previous TTL (or initial-TTL) or setting a new TTL value.

NOTE: Generally, initial-TTL is taken from system headers. If it is not passed from the ingress side.

This decision is made according to the TTL-Model:

- Uniform – Keep the previous TTL
- Pipe – Set a new TTL

For the Pipe model, the TTL parameters are provided to decide the new TTL value. These parameters are derived according to the OutLIF or Out-ETH-RIF entry.

Unlike other QoS sections, there is no QoS profile as part of the egress TTL process.

10.12.1 SOC Properties

None

10.12.2 Configuration Flow

TTL-Model Configuration: The options in the following table can be configured.

Object	API	TTL-Model
L3 interface	`bcm_l3_intf_create`	Always Uniform
MPLS-tunnel	`bcm_mpls_tunnel_initiator_create`	`label_array[0].egress_qos_model.egress_ttl = bcmQosEgressModelPipeMyNameSpace` Set to PIPE, otherwise Uniform
PWE	`bcm_mpls_port_add`	`egress_label.egress_qos_model.egress_ttl = bcmQosEgressModelPipeMyNameSpace` Set to PIPE, otherwise Uniform
IP-tunnel	`bcm_tunnel_initiator_create`	`tunnel.egress_qos_model.egress_ttl = bcmQosEgressModelUniform` Set to Uniform, otherwise Pipe

TTL Pipe Properties: If TTL-Model is Pipe, the parameters in the following table decide the new TTL value (for both marking and remarking).

Object	API	New TTL Value
MPLS-tunnel	`bcm_mpls_tunnel_initiator_create`	`label_array[0].ttl`
PWE	`bcm_mpls_port_add`	`egress_label.ttl`
IP-tunnel	`bcm_tunnel_initiator_create`	`tunnel.ttl`

10.12.3 Shell Commands

None

10.12.4 Application Reference

Example of that egress TTL of MPLS tunnel is pipe mode

- **Type:** CINT reference
- **Path**: `$SDK/src/examples/sand/cint_sand_mpls_tunnel_initiator.c`

10.13 Egress VLAN Translation VLAN-PCP QOS

As part of the Egress VLAN translation mechanism (EVE), it is possible to edit the PCP-DEI of the Ethernet VLAN headers. QoS mapping of egress VLAN translation is done at the egress device according to the Out-AC (Egress VLAN-Port) or ARP+ AC QoS profile. The QoS profile for VLAN translation is the same as the object's Egress Remark QOS-profile. At the egress device, the resulting `NWK_QOS` values are taken as PCP, DEI (up to two tags). For bridge services, EVE can select the updated PCP, DEI as final output.

For more information about egress VLAN translation, see Section 21.7, VLAN-Translation AC-LIF.

10.13.1 SOC Properties

None

10.13.2 Configuration Flow

See Section 10.10, Egress Marking, Remarking, and Inheritance.

10.13.3 Shell Commands

None

10.13.4 Application Reference

Example of egress VLAN translation PCP mapping:

- **Type:** CINT reference
- **Path:** `$SDK/src/examples/dnx/cint_qos_l2_vlan_edit.c`

10.14 Egress CoS Remapping

To change the packet TC and DP, which would not affect the egress TM queuing, the egress PMF can provide the egress CoS profile to the ETPP without TM queuing changes. The ETPP will get new TC, DP, and NWK-QOS through CoS profile mapping. Default CoS profile 0 is reserved and will not update TC, DP, and `c`.

10.14.1 SOC Properties

None

10.14.2 Configuration Flow

1. **Create egress COS profile**: Allocate a new COS-profile by calling:
   `bcm_qos_map_create(unit, flags, &map_id);`
   - `flags` – `BCM_QOS_MAP_ENCAP_INTPRI_COLOR` | `BCM_QOS_MAP_EGRESS`. `flags` can also receive `BCM_QOS_MAP_WITH_ID`, which allows the user to choose the ID allocated, and the SDK adds a prefix to encode the ID.
2. Add COS mapping: New COS mapping by calling
   `bcm_qos_map_add(unit, flags, &in_map, cos_map_id);`
   - `cos_map_id` – The encoded COS-Profile ID
   - `in_map.int_pri` – The new mapped TC
   - `in_map.color` – The new mapped DP
   - `in_map.dscp` – The new mapped NWK_QOS
   - `flags` – `BCM_QOS_MAP_ENCAP_INTPRI_COLOR`

NOTE:

- If `in_map.int_pri` is larger than `BCM_COS_MAX`, the field in the mapping is invalid, and FTMH_TC will not be updated.
- If `in_map.color` is `bcmColorPreserve` or above, the field is invalid in the mapping, and FTMH_DP will not be updated.
- If `in_map.dscp` is larger than 255, the `NWK_QOS` field is invalid in the mapping and will not be updated.
- In IOP mode, the new mapped `NWK_QOS` cannot be selected. `NWK_QOS` always comes from FHEI if FHEI exists and the forward code is IP or MPLS, otherwise, `NWK_QOS` is 0. That is to say, this mapping cannot update `NWK_QOS` in IOP mode.

3. Set the CoS profile by using ePMF. Add a field entry with action type – `bcmFieldActionQosMapIdNew`.
   For more information see Chapter 11, Field Processor.

10.14.3 Shell Commands

None

10.14.4 Application Reference

Example of application to configure egress CoS profile and mapping

- **Type**: CINT reference
- **Path**: `$SDK/src/examples/dnx/qos/cint_dnx_qos_egress_cos.c`

10.15 MPLS Ingress Swap QoS

For an MPLS ingress swap, the swapping of MPLS encapsulation information is built at the ingress and passed to the egress by system headers. Then, it is translated to an encapsulation entry for usage later in the encapsulation stage.

QoS attributes (such as remark-profile, TTL model, and TTL value) can be retrieved from the entry directly or indirectly (swap action non-EEDB). These attributes can be configured separately by the QoS control APIs described in this section.

NOTE:

- For the exact number of remark-profiles supported, see the MPLS Ingress Swap Remark-profile information in Appendix A, Device Family Differences.
- The TTL model and TTL value configurations are global. MPLS swapping and pushing services using the same encapsulation entry share the same TTL model and TTL value.

For more information, see Section 23.7, Remark Profile for SWAP Actions Coming Out of Ingress (ILM, FEC).

10.15.1 SOC Properties

None

10.15.2 Configuration Flow

For multi-swap command device support, to specify the swap-remark-profile, use `bcm_mpls_tunnel_switch_create` and `info.qos_map_id` as the swap remark profile (or use `bcm_l3_egress_create` and `egr.qos_map_id`).

NOTE: For the BCM88690 and BCM88800 devices, set to 0.

For single-swap command device support, set the global QoS profile for MPLS ingress swap by calling the following API:

```
bcm_qos_control_set(int unit, uint32 flags, bcm_qos_control_type_t type, int arg)
```

- `type` – `bcmQosControlMplsIngressSwapRemarkProfile`
- `arg` – QoS profile.

To set the global TTL model and TTL value, call the following API:

```
bcm_qos_control_set(int unit, uint32 flags, bcm_qos_control_type_t type, int arg)
```

- For the TTL model
  - `type` – `bcmQosControlMplsIngressSwapTtlModel`
  - `arg`:
    - `bcmQosEgressModelUniform` – Inheriting the TTL (default value).
    - `bcmQosEgressModelPipeMyNameSpace` – Using the configured TTL.
- For the TTL value
  - `type` – `bcmQosControlMplsIngressSwapTtl`
  - `arg` – Integer between 0 and 255.

NOTE: The information is used by MPLS encapsulation (MPLS swap can never be a Pipe model).

10.15.3 Shell Commands

None

10.15.4 Application Reference

Example of MPLS Ingress Swap usage

- **Type:** CINT reference
- **Path:** `$SDK/projects/branch/src/examples/sand/cint_sand_mpls_qos.c`
  `mpls_ingress_swap_remark_profile_set(int unit, int remark_profile)`

10.16 Ingress Explicit Null Label QoS Attributes

The explicit NULL label is terminated at ingress. QoS attributes, such as QoS profile, PHB model, remark model, and TTL model, can be globally configured on the explicit NULL label for variable QoS selection.

10.16.1 SOC Properties

None

10.16.2 Configuration Flow

Explicit NULL label QoS attributes can be configured with the `bcm_qos_control_set (int unit, uint32 flags, bcm_qos_control_type_t type, int arg)` API.

- `flags` – `BCM_QOS_MAP_IPV4` or `BCM_QOS_MAP_IPV6`
- `type`:
  - `bcmQosControlMplsExplicitNullIngressQosProfile`
  - `bcmQosControlMplsExplicitNullIngressPhbModel`
  - `bcmQosControlMplsExplicitNullIngressRemarkModel`
  - `bcmQosControlMplsExplicitNullIngressTtlModel`
- `arg`:
  - QoS profile – `qos_map_id`
  - QoS model – `bcmQosIngressModelShortpipe`, `bcmQosIngressModelPipe`, `bcmQosIngressModelUniform`

10.16.3 Shell Commands

None

10.16.4 Application Reference

Ingress explicit NULL label QoS attribute examples

- **Description:** Example of an IPv4 explicit NULL label pipe model for PHB.
- **Type:** CINT reference code
- **Path:** `$SDK/src/examples/sand/cint_sand_mpls_qos.c`

10.17 Ingress ELI Label QoS Attributes

The ELI and EL label is terminated at ingress. QoS attributes, such as QoS profile, PHB model, remark model, and TTL model, can be globally configured on the ELI label for variable QoS selection.

10.17.1 SOC Properties

None.

10.17.2 Configuration Flow

Explicit Null label QOS attributes can be configured with API `bcm_qos_control_set(int unit, uint32 flags, bcm_qos_control_type_t type, int arg)`.

- `type`:
  - `bcmQosControlMplsELIIngressQosProfile`
  - `bcmQosControlMplsELIIngressPhbModel`
  - `bcmQosControlMplsELIIngressRemarkModel`
  - `bcmQosControlMplsELIIngressTtlModel`
- `arg` – `qos_map_id` for QoS profile, or `bcmQosIngressModelShortpipe`, `bcmQosIngressModelPipe`, `bcmQosIngressModelUniform` for QoS model.

10.17.3 Shell Commands

None.

10.17.4 Application Reference

None.

10.18 API Descriptions

10.18.1 bcm_qos_*

API Name	Highlights
`bcm_qos_map_create()`	Software profile allocations according to flags. **NOTE:** Without the flag `BCM_QOS_MAP_WITH_ID`, it is not guaranteed to get a certain profile ID.
`bcm_qos_map_destroy()`	Destroy the software QoS profile.
`bcm_qos_map_add()`	Add map entry, related to specific QoS application.
`bcm_qos_map_multi_get()`	Get multiple map entries or the number of entries for a given map ID and mapping flag. If the flags include `BCM_QOS_MAP_OPCODE`, get the opcode for a given map ID and application flag. Map ID should be mapped to the opcode to get application QoS mapping entries. In the egress, if map ID is a mapped profile without the `BCM_QOS_MAP_OPCODE` flag, get QoS inheritance mapping for the next layer.
`bcm_qos_map_delete()`	Deletes map entry.

API Name	Highlights
`bcm_qos_port_map_set()`	Set the QoS profile to the gport, sets ingress and egress map IDs, each of them should have the appropriate prefix set according to its type. If QoS profile 0 must be set to the gport, first call `bcm_qos_map_id_get_by_profile()` to get the encoded map ID for base profile ID 0.
`bcm_qos_port_map_type_get()`	Get the QoS profile from the gport according to flags.
`bcm_qos_port_map_get()`	Get the ingress and egress QoS profile from the gport.
`bcm_qos_multi_get()`	If `array_size` is zero, get number of allocated QoS profile, otherwise get allocated QoS profile and creation flags.
`bcm_qos_map_id_get_by_profile()`	Get encoded map ID base for the QoS profile.

Chapter 11: Field Processor

11.1 Introduction

The Field Processor (FP) module provides APIs to the Programmable Mapping and Filtering (PMF) blocks (in the ingress and egress). It enables the implementation of various networking applications such as ACL and policy based routing, as well as proprietary features.

The FP has four instances divided into two categories:

- **Ingress processor**

  The ingress processor includes three FP instances: iPMF1, iPMF2, and iPMF3.

  Each instance is a generic field processor that performs lookups based on any of the ingress packet variables resolved in the current stage, and on packet header fields. Each instance can override a set of signals as defined in the action section.

  Each stage can use the results of the previous stage:
  - In iPMF-1/2 the lookup is performed after the forwarding stages but before the FEC resolution stage.
  - In iPMF-3 the lookup is performed after the FEC resolution stage.
- **Egress processor**

  The egress processor includes one FP instance: ePMF.

  The ePMF is a generic field processor that performs lookups based on any of the predefined egress packet variables, system header fields, or networking header fields.

  This instance overrides the set of signals defined in the action section.

  The egress processor supports only two TCAM lookups. It also has an action to point to the ACE extension.

11.1.1 Field Groups

The FP maintains several databases, called Field-Groups (FG). An FG generates actions to a packet according to its qualifier values.

11.1.1.1 Qualifiers

Packet qualifiers are fields from packet headers or various variables derived from the packets through the processing pipe. For example: Source port, TC, DP, and VSI (this is known as metadata).

Qualifiers can also be:

- A range of L4 ports or OutLIF IDs.
- Derived from other sources such as a compare operation result, cascaded-data, constants and values.

11.1.1.2 Preselectors

Per packet, context is selected based on preselectors. Preselectors are a set of qualifiers that determine the context. The context determines which FGs are looked-up, and how to process their actions. Any given FG can be attached to more than one context.

11.1.1.3 Lookups

Per <FG, Context> a dedicated lookup key is created. Each lookup is orthogonal to the others.

The result of the lookup is parsed to action values according to the action set of that field group. The resulting action values (such as destination ports, Traffic Class, DP, and so on) override the values obtained from the packet processing pipe. As a result, a field group can implement filters and forwarding.

For a TCAM lookup, each FG may contain several matching entries. The entry that gets selected is the one with the highest priority. If two entries with the same priority match, one of them is selected arbitrarily.

When an action (such as a destination port) has multiple matching entries in multiple FGs, the resolution is according to the action priority. Only the action with the highest priority is taken into account.

11.1.1.4 Actions

Actions are values associated with packet processing (PP). In the FP (that is, PMF block) packets are classified based on predefined protocol fields from Layer 2 to Layer 7. Packets are also classified according to user definitions. Based on these classifications, the following actions can be applied to a packet:

- Discarding the packet.
- Sending the packet to the CPU.
- Sending the packet to a mirror port.
- Updating destination port or FEC.
- Modify the CoS attributes of the packet.
- Associate the packet with a meter/counter instance.

The field processor FG has four main methods for retrieving actions:

- TCAM Lookup
- Direct-Extraction
- Exact Match

11.1.1.4.1 TCAM Lookup

In TCAM Lookup, the user specifies the values of the qualifying attributes and the action values.

For example: A policy-based routing FG, which updates the forwarding FEC value according to {DSCP, DIP, In-RIF}. An entry in this FG can map all packets matching {DSCP = 7, DIP = 1.2.2.3, In-RIF = 3} to FEC = 9.

Each entry in a FG must specify:

- Value and mask for each qualifying attribute, as defined for the FG. By default, the qualifier masks of a new entry are set to ignore the qualifier value.
- The actions to be applied to qualified packets. By default, the entry does not perform any action. The action is performed only if the user has inserted the action value.
- A priority, used to resolve conflicts between entries in the FG.

11.1.1.4.2 Direct-Extraction

Direct-Extraction provides an optimized method to configure a rule that maps a range of certain actions (such as meters) according to the qualifying attributes. Although such mappings are also possible using TCAM FG, Direct-Extraction can save a lot of TCAM resources.

In Direct-Extraction, an entry is an instruction indicating how to calculate the action values from the values extracted from the qualifying attributes. For example, a logical FG can be defined to attach packets to meters according to the incoming Attachment Circuit. A single entry in this FG can attach Meter-IDs 2K-3K to packets that come on In-ACs 0-1K respectively. This mapping is implemented as a Direct-Extraction entry as follows:

- The qualifier of the entry is In-AC.
- The action instructions are:
  - Extract bits [9:0] from the In-AC.
  - Add 2K to the extracted value.
  - Set the result as the Meter-ID.

11.1.1.4.3 Exact Match

In Extra Exact Match (EXEM), a lookup is performed without a mask, unlike for TCAM, and the entire key must match (although the key size may vary).

11.1.2 Application Configuration Checklist

- Field Groups
- Qualifiers and Actions
- Range qualifiers
- Contexts
- Direct Extraction
- Constant Field Group
- FEM actions
- Cascading
- Compare
- Hash
- ACE – Egress PMF extension
- EXEM (Extra Exact Match) Lookup
- System Headers
- LIF and Port Profiles
- Diagnostics

11.2 Field Groups

A field group defines a logical database that is comprised of:

- A set of qualifiers which construct a key
- A set of actions to be performed
- A set of ACL rules (entries)

The API `bcm_field_group_add()` does the following:

- Defines the Field Group qualifiers and actions
- Manages the TCAM/MDB TCAM

11.2.1 SOC Properties

N/A

11.2.2 Configuration Flow

To add a new field group, call:

```
bcm_field_group_add(unit, flags, &fg_info, &fg_id);
```

For a detailed description, see Section 11.28.1.1, bcm_field_group_add().

11.2.3 Application Reference

Direct extraction example:

- **Type**: Default Application Reference
- **Path**: `$SDK/src/appl/reference/dnx/appl_ref_itmh_pph_init.c`

TCAM Lookup Example:

- **Type:** Cint
- **Path:** `$SDK/src/examples/dnx/field/cint_field_attach.c`

11.3 Contexts

In each stage the packet is mapped to a context. In previous devices this was called PMF-Program.

Contexts are used for:

- Selecting the FGs relevant for the packet
- Extracting the qualifiers from various inputs

11.3.1 Context Selection (Presel)

A context is selected by a TCAM table which maps traffic to a single context-ID. Each entry in the table defines the set of qualifiers (preselectors).

When there is a match the context is used. If there is no match a default context is used. By default, all packets are mapped to a default context.

11.3.2 Adding Contexts

There are two reasons for adding new contexts.

- **Support for multiple packet types**

  The context defines the way to extract the qualifiers. Multiple contexts must be defined when extracting qualifiers from different input formats.
- **Mapping different packets to different FGs**

  This allows searching only the relevant FGs. There is a limited number of lookups per packet, therefore searching only the relevant FGs increases the number of effective lookups per packet.

11.3.3 Context Searching

Different contexts can generate different keys for the same field group. For example, for the same FG with the qualifier **Dest_IP Address**, the following three contexts can be searched by a single TCAM entry:

- Context A can search the inner DIP in the FG
- Context B can search the outer DIP in the FG
- Context C can search the DIP in the forwarding header in the FG

iPMF2 Stage Context

There are two types of iPMF2 contexts:

- Initial iPMF2 contexts:
  - Context in use when there is no hit in iPMF2 context selection TCAM
  - When the user creates an iPMF1 context, the SDK also creates an iPMF2 context with the same ID.
- Cascaded iPMF2 contexts:
  - Context in use when there is a hit in the iPMF2 context selection TCAM
  - Generated by calling `bcm_field_context_create()`.
    - stage=iPMF2
    - `.context_mode_p` → `context_ipmf2_mode.cascaded_from` is an iPMF1 context-ID that is cascaded from by the currently created context. The context is used in iPMF1. (iPMF2 is an extension of the iPMF1 context, and therefore, the iPMF1-context-ID must be provided.)
  - Use `bcm_field_presel_set()` to map traffic to iPMF2 according to iPMF1 FG results

For an example of the creation of a new context, see Section 11.3.6, Configuration Flow.

11.3.4 Attach Field Group to a Context

After adding new Field Groups (FGs) by calling `bcm_field_group_add()`, those FG should now be triggered by some context to perform those lookups. To achieve that, `bcm_field_group_context_attach()` was created and allows customers to build lookup keys differently per context.

For example, in one context, `Src_IPv4` can be taken from the inner IP, and in another context, `Src_IPv4` can be taken from the outer IP.

11.3.5 SOC Properties

N/A

11.3.6 Configuration Flow

1. Create a new context.

   `bcm_field_context_create()` – Create a new FP context
   - Allocate context ID. WITH_ID / without ID
   - Context is created per Field Stage: iPMF1/2/3, ePMF

   For more information, see Section 11.28.2.1, bcm_field_context_create().

2. Attach FGs.

   Call `bcm_field_group_context_attach()`.

   For more information, see Section 11.28, API Descriptions.

NOTE: For TCAM field groups, it is best practice to add entries to the field group after attaching the field group to the context. This is because different keys on one context cannot access the same TCAM banks, so adding entries after attaching allows a more efficient TCAM bank allocation.

3. Map traffic to the created context (presel set).

   Call `bcm_field_presel_set()`. Map traffic to the context (Default context is chosen when there is no hit).

   Indicate the presel entry ID:
   - stage – Context selection stage
   - presel-Id – The line inside the context selection CAM table (Lower index means higher priority)

   For more information, see Section 11.28.3.1, bcm_field_presel_set().

11.3.7 Application Reference

Examples of applications for adding a context can be found in the ITMH-programmable reference application.

- **Type:** Default Application Reference
- **Path:** `$SDK/src/appl/reference/dnx/appl_ref_itmh_pph_init.c`
- **Function:** `appl_dnx_itmh_init()`

11.4 Qualifiers

Qualifiers are classifiers for a packet lookup. A set of qualifiers comprise a key.

There are a few qualifier input types:

- Metadata – Decisions and classifications done by the pipeline
- Packet layers – Examples include ETH, IPv4, MPLS, and ITMH
- Layer-Records – Parser information about the packet layer
- Cascading – Using the results of iPMF1 as qualifiers for iPMF2 lookup. As there is no action resolution between iPMF1 and iPMF2:
  - The qual value prior to iPMF1 is selected by the metadata
  - The qual value after iPMF1 is selected by Cascading
- Const values – To put a constant value on the key.

11.4.1 Usage Flow

1. Create a FG and specify the qualifier IDs for example DIP, OutLIF.
2. Attach the FG to the context. Specify the extraction information, for example:
   - DIP: From second layer / third layer / forwarding layer
   - OutLIF: Metadata / cascaded FG
3. Add an entry to the FG. Specify the qualifier value for example:
   - DIP: 10.1.2.200
   - OutLIF: 300

11.4.2 User-Defined Qualifiers

It is possible to create a qualifier with a user-defined size, offset, and name (for the calling sequence information, see Section 11.4.4, Configuration Flow). Typical usages include:

- Qualifiers that only use a subset of a predefined qualifiers, for example when the key to a FG only requires bits 10:2 of a LIF.

  To retrieve information for specific predefined qualifiers, use `bcm_field_qualifier_info_get()`. This API will provide parameters, such as size and offset, in the metadata.
- Const Qualifiers: Adding constant bits to a FG key
- To copy data from a user-defined offset, for example, to add a user-proprietary information to a FG key
- When there are two qualifiers from the same type (for example, when two DIPs exists), a user-qualifier is needed

11.4.3 Raw Qualifiers

On some qualifiers, the SDK performs mapping from the BCM value to the DBAL or hardware value. For example, with `bcmFieldQualifySrcPort`, the BCM input is GPORT, which is later mapped to the hardware value.

NOTE: When the input GPORT is a member of a LAG, it is converted to a system port aggregate (SPA) that is encoded with the corresponding LAG-group and LAG-member-ID.

Another example is `bcmFieldQualifyForwardingType`. The BCM input is the type `bcm_field_layer_type_t`, which is later mapped to the DBAL enum.

This type of qualifier has an additional BCM qualifier that is a *raw* qualifier. On a raw qualifier, the SDK does not perform any mapping, and it is expected that the input on the BCM layer is already mapped to the DBAL or hardware.

User-defined qualifiers are always expected if they are raw qualifiers, so the following API handles this type of mapping:
`bcm_field_qualifier_value_map()`

The following section provides additional details.

11.4.4 Configuration Flow

To create a new user-defined qualifier:

- Call `bcm_field_qualifier_create()` to generate the qualifier, with Input parameters:
  - Flags – `BCM_FIELD_WITH_ID`. Used to add a qualifier with ID.
  - Name – Optional. If set, it should be unique per qualifier.
  - Stage
  - Size

  Input/Output – (Depends if WITH_ID was set)
  - qualifier Id: Use the qualifier ID when adding the qualifier to FGs
- When attaching to context: Provide offset to specify the location of the qualifier

If the qualifier needs value mapping, use the `bcm_field_qualifier_value_map()` API to map the required value to the corresponding HW value before inserting it to the HW:

Input parameters:

- Stage – The stage the qualifier is being used in
- Qualifier – The predefined qualifier to map the value for
- `bcm_value` – Value to map buffer (maximum size 160b)

Output parameters:

- `hw_value` – Mapped HW value (to be inserted to HW)

11.4.5 Application Reference

Example of application using user-defined qualifier

- **Description**: Shows an example for configuration of user qualifier on EtherType, which is relevant for untagged and tagged Ethernet packets.
- **Type**: Cint
- **Path**: `$SDK/src/examples/dnx/field/cint_field_user_qual_ethertype.c`

11.4.6 Predefined Qualifiers

For a list of Predefined qualifiers, see Section 11.31.2, Qualifiers List.

11.4.7 Range Qualifiers

A range qualifier is based on a range of configurable packet attributes.

Each range is configured using minimum and maximum values matched inclusively (Min ≤ Value ≤ Max). The result indicates if the packet fits into the range. The results can then be used for the key construction or for the program selection.

Supported Ranges (minimum and maximum values, inclusive):

- L4 port ranges, or L4 Ops –There are 24 range filters defined for both ingress and egress ports (24 profiles for each). The result is a 24b' bitmap. A *hit* for a port range occurs only if the packet parameters fit both ingress and egress ranges. The bitmap will include all such hits.
- Packet size ranges – There are three filters in iPMF1. The values are in byte resolution. A *hit* for a size range occurs if the packet size fits the range. If the packet size ≥ 144B, it is treated as 144B; otherwise, it qualifies on the real packet size. The result is 4b', where the first matched range is returned (used as priority). If none of the ranges fit, the result value will be 3.
- OutLIF ranges (three ranges total, and each range can be applied for OutLIF0/OutLIF1) – Used for preselection only.

NOTE: The qualifier for L4 port ranges (`bcmFieldQualifyRangeCheck`) acts differently than the qualifiers for packet size and OutLIF ranges (`bcmFieldQualifyPacketLengthRangeCheck` and `bcmFieldQualifyVPortRangeCheck` respectively). The qualifier for L4 port ranges qualifies on a bitmap of all the range IDs, while qualifiers for the other range types provide the first range ID that matches the value. Consequently, all L4 port ranges are initialized by default to a range that accepts all values (0-65535) while the other ranges are initialized by default to have their maximum value smaller than their minimum, thus accepting no value.

11.4.8 Range Qualifiers Extension

NOTE: This feature is supported only on the BCM88830 device.

Additional features have been added to range qualifiers of L4 ports. The following capabilities have been added:

- Additional range qualifiers in addition to L4 ports:
  - In TTL
  - Packet size
  - UserQualifier_1_low (16 LSB)
  - UserQualifier_1_high (16 MSB)

  - UserQualifier_2_low (16 LSB)
  - UserQualifier_2_high (16 MSB)

  `bcm_field_range_type_config_set()` configures `UserQualifiers`.

  The same `bcmFieldQualifyRangeCheck` qualifier is used (it reflects only the 24 LSB output of the first compare chunk).

- Mux to select which range qualifier to check (out of those mentioned previously). Up to four different qualifiers can be selected.
  - `bcm_switch_control_indexed_set()`
    - Key type: `bcmSwitchRangeTypeSelect`
    - Key value: 0 to 3
    - Value: `bcmFieldRangeTypeXXX`
- Operator performed between the selected range IDS, as explained in Section 11.4.7, Range Qualifiers. The range bit is set to 1 only when both L4src and L4dst ranges match. That is, the operator is set to AND (&).
  - `bcm_switch_control_set()`
    - Key: `bcmSwitchRangeOperator`
    - Value: `bcm_switch_range_operator_t` (all enum values)
- Encoders that pick the best ranges to compare. The encoder output is the smallest hit index when a hit occurs along with 1 at the MSB.

  The MSB changes according to the number of chosen ranges. For example, when one range (32b) is chosen, the hit indication MSB is at bit 5. When two ranges are chosen, it is at bit 6. The hit indication MSB is at bit 7 when three or four ranges are chosen.

  If no hit occurred for the chosen ranges, the output of the encoder is 0.

  The output of the encoders can be accessed with the qualifiers `bcmFieldQualifyRangeFirstHit0...3`.

  To set the encoders, use the following API:
  - `bcm_switch_control_indexed_set()`
    - Key type: `bcmSwitchRangeResultMap`
    - Key value: 0 to 3 (the four encoders have a 1:1 match to the four qualifiers)
    - Value: `bcm_switch_range_result_map_t` (all enum values)

11.4.9 SOC Properties

N/A

11.4.10 Configuration Flow

1. Create a range using the API `bcm_field_range_set()`

   Indicate the following items:
   - Stage in which the range should be applied. For different range types, not all stages are applicable (for example, range type Packet Header Size only applies for iPMF-1 and iPMF-2 stages).
   - Range-ID
   - Range-info:
     - Range type: the type of range to use, which is one of the following:
       - L4Port Range (24 ranges):

         The API should be called once per type on the same range.
         - `bcmFieldRangeTypeL4SrcPort`

  - bcmFieldRangeTypeL4DestPort
  - bcmFieldRangeTypeOutVport
  - bcmFieldRangeTypePacketHeaderSize

  The BCM88830 supports additional range types. For more information, see the check API section of bcm_field_range_set().
- min_val
- max_val

NOTE: Minimum or maximum are not mandatory. If one is not indicated, the absolute minimum or maximum is used. If neither is indicated, the default configuration is used.

2. Create a field group using a predefined qualifier or user-defined qualifier based on the desired range-ID.
   a. If using a predefined qualifier, include the following BCM qualifier in the qualifier array of the field group:
      - bcmFieldQualifyRangeCheck for L4 ports. This qualifier is a bitmap.
      - bcmFieldQualifyPacketLengthRangeCheck for packet Header Size.
   b. If using a user-defined qualifier, first create a qualifier using the bcm_field_qualfier_create API and indicate its size. For more information, see the qualifiers and actions sections. When attaching the field group for the context, indicate the offset of the qualifier in the attach information.
   c. Create entries, setting the qualifier value that corresponds to the ranges to be checked. For example, to classify upon ranges 4 to 7, the value 0x10 should be set as the range bitmap.
3. If the range qualifier is used for preselection, use the bcm_field_presel_set API, with the bcmFieldQualifyVPortRangeCheck indication in the presel_entry_info. The expected value of the range ID should be supplied in the qualifier.
4. The range can be retrieved using bcm_field_range_info_get() API.

11.4.11 Application Reference

Example of using range qualifiers in key construction.

- **Type:** CINT example
- **Path:** $SDK/src/examples/dnx/field/cint_field_range_qual.c

Example with new range qualifier types.

- **Type**: CINT example
- **Path**: $SDK/src/examples/dnx/field/cint_field_range_qual_ext.c

NOTE: The feature is available only for the BCM88830 device.

11.5 Actions

Actions are the result of a lookup, which changes decisions about the packet by modifying pipeline signals, such as: destination, in/out LIF, TC, DP, and so on. These values often relate to PP data.

There are two types of actions:

- Predefined actions
- User-defined actions

11.5.1 Predefined Actions

A set of actions exists with fixed size and meaning, i.e., which signal(s)/headers to update. For the list of actions per stage, see Section 11.31.3, Actions List.

11.5.2 User-Defined Actions

A new action ID can be defined, with desired size based on predefined action, unless the action is void, which holds a place in the result payload. There are two typical uses of user-defined actions:

- Void Actions: Results that do not translate to actions by the field group. For example:
  - Cascading: iPMF1 TCAM result that will only be used as qualifier to iPMF2 FG
  - Results that require FEM
- Actions with prefix. For example
  - 20 bit OutLIF action in which 12 bits taken from the TCAM payload or DE key and the eight MSB are set to a const value.
  - Actions with a prefix value of zero can be used to save space on the payload when all bits are not needed on the action, and the use of a prefix value of zero does not consume any extra resources.
  - Actions with a prefix value that is nonzero cannot be used by a FEM machine (for more information, see Section 11.16, Field Extraction Macro).
  - Actions with zero size are only applicable in Field Group Type `bcmFieldGroupTypeConst` (see Section 11.22, Constant Field Group). In all other field group types each action must be at least 1 bit.

11.6 Raw Action

A *raw* action has the same meaning as a raw qualifier. See Section 11.4.3, Raw Qualifiers. Action has its own mapping function: `bcm_field_action_value_map()`.

11.6.1 Action Priority

When attaching FG to a context, the user may provide ‘Action priority’. The action priority is defined per context and action.

Action priority is relevant only under the following conditions:

- Several FGs occur in the same stage. Unless the stage is iPMF1/iPMF2 because for iPMF1, the action resolution occurs only after iPMF2, and both stages share the same action resources.
- Several FGs generate the same action type
- Several FGs are attached to the same context

When priority is not mandatory, it is better to use Don’t care. For example, if for the configured context and stage, only one FG can affect the packet’s TC, the action priority field should be *don’t care* (`BCM_FIELD_ACTION_DONT_CARE`).

The FES ID with the higher number, will win when the same action is generated by several FGs in the same context and stage.

Number of Action macros and priority per stage:

- iPMF1,2: There are 32 EFESs and 16 FEMs in four arrays.
  - FEMs 15–8 (Highest priority)
  - EFESs 31–16
  - FEMs 7–0
  - EFESs 15–0 (Lowest priority)
- ePMF 1 array of 12 EFESs
- iPMF3 1 array of 16 EFESs

Action priority can either be managed by the SDK or by the user by using the following MACROs:

- `BCM_FIELD_ACTION_POSITION(array_id, position)` is used to select a specific FES position.
- `BCM_FIELD_ACTION_PRIORITY(array_id, priority)` will let the SDK select a FES according to action priority (but still the user selects the FES array).
- `BCM_FIELD_ACTION_PRIORITY_DONT_CARE` indicates that any FES can execute the action (from any FES array).

When managing FES ID, user needs to be aware of the following hardware restrictions:

- A single context cannot use the same FES ID twice.
- There are more contexts than FES-programs.
- There are three prefixes per FES ID.

FES ID Management by Position

Hardware restrictions when managing FES ID:

- There are more contexts than FES-programs.
- There is a limited number of non-zero prefixes per FES ID.

For each <PMF stage, context, action type>:

- The user can either provide position or priority, but not both
- In other words, the SDK will return an error, if two FGs are:
  - On the same PMF stage (iPMF1/2 are considered to be the same stage here).
  - Attached to the same context (cascaded contexts included).
  - Both generate the same action.
  - One of the actions’ priority is managed by `BCM_FIELD_ACTION_POSITION` and the other is managed by `BCM_FIELD_ACTION_PRIORITY`

- `BCM_FIELD_ACTION_POSITION()` must be used when using the Invalidate Next action (see Invalidate Next Action).
- Only `BCM_FIELD_ACTION_POSITION()` can place an action immediately following an Invalidate Next action (thus allowing the action to be affected by it).

FES ID Management by Priority

Priority-based management still require the user to provide array-id

- For iPMF3 and ePMF, Array-ID is always 0
- For iPMF1 and iPMF2, Array, the ID is 0-3
- `BCM_FIELD_ACTION_PRIORITY_DONT_CARE` doesn't require array ID

The action priority EFES algorithm:

- Try to do EFES sharing when possible
- Look for valid EFES IDs according to:
  - Priorities of other actions with the same action type
  - Hardware constraints: Availability of FES-programs, prefixes (masks), and same-contexts actions
- Reshuffle other actions upon need
  - Only `BCM_FIELD_ACTION_PRIORITY_DONT_CARE` actions can move to other arrays
  - Reshuffling does not affect traffic.

The usage of `BCM_FIELD_ACTION_POSITION(array_id, position)` is:

- ePMF `BCM_FIELD_ACTION_POSITION(array = 0, 11>=position>=0)`
- iPMF3 `BCM_FIELD_ACTION_POSITION(array = 0, 15>=position>=0)`
- iPMF1&2
  - FEMs 15–8 `BCM_FIELD_ACTION_POSITION(array = 3, 7>=position>=0)`
  - EFESs 31–16 `BCM_FIELD_ACTION_POSITION(array = 2, 15>=position>=0)`
  - FEMs 7–0 `BCM_FIELD_ACTION_POSITION(array = 1, 7>=position>=0)`
  - EFESs 15–0 `BCM_FIELD_ACTION_POSITION(array = 0, 15>=position>=0)`

`BCM_FIELD_ACTION_PRIORITY(array_id, priority)`

- ePMF, iPMF3 `BCM_FIELD_ACTION_PRIORITY(array = 0, 16bit priority)`
- iPMF1&2
  - EFESs 31–16 `BCM_FIELD_ACTION_POSITION(array = 2, 16bit priority)`
  - EFESs 15–0 `BCM_FIELD_ACTION_POSITION(array = 0, 16bit priority)`
  - FEMs can only be managed by `BCM_FIELD_ACTION_POSITION()`. For details, see Section 11.16, Field Extraction Macro.

Invalidate Next Action

The invalidate next action (`bcmFieldActionInvalidNext`) makes any action committed by the following EFES or FEM invalid.

The invalidate next action can only be placed in an EFES with a priority that indicated their exact placement (`BCM_FIELD_ACTION_POSITION()`, not `BCM_FIELD_ACTION_PRIORITY()`), and an EFES immediately following an EFES containing an invalidate next action (for a specific context) can only have an action with a priority that indicates an exact placement. In FEM actions can only be placed using exact placement anyway.

In iPMF1/2, where both EFES arrays and FEM arrays exist, if the invalidate next action is valid in the last EFES in the FES array, it invalidates the first FEM in the following FEM array. If it happens in the last FEM in the first FEM array, it invalidates the first EFES in the second EFES array. The first EFES in the second EFES array can be allocated for actions with numerical priority (`BCM_FIELD_ACTION_PRIORITY()`) without regard to whether the last FEM in the first FEM array has an invalidate next action or not. Consequently, every time an invalidate next action is configured in the last FEM of a FEM array, the first EFES in the following EFES array should be filled using exact placement priority.

An important capability of invalidate next is that it allows one field group to disable the actions of another. One use case, as done when using strict unicast RPF with ECMPs, is to allow default actions for an exact match lookup to be performed. A Const FG is created that performs certain actions, and each exact match entry performs invalidate next actions on the Const FG's actions. The Const FG's actions take place only if none of the exact match entries are hit.

Invalidate Action by Priority

Using the `BCM_FIELD_ACTION_INVALIDATE` priority invalidates an action, effectively turning it into a void action for a specific context only. This allows the use of the same field group for multiple contexts, even if some actions should not occur for some contexts. For an example, see Section 11.6.4, Application Reference.

Adding Actions after Creating the Field Group

It is possible to add actions beyond those that were created using `bcm_field_group_add()`. One option is to add a FEM using the `bcm_field_fem_action_add()` API, as described in Section 11.16, Field Extraction Macro. Another option is to add an EFES using the `bcm_field_efes_action_add()` API as described in Section 11.17, Enhanced Field Extraction Shifter.

11.6.2 SOC Properties

N/A

11.6.3 Configuration Flow

To create a new user-defined Action, call `bcm_field_action_create()` API to generate the new action with:

Input parameters:
- Flags – WITH_ID (if not set, ID allocated by SDK)
- Name – String (optional)
- Stage – FP stage in which the action is defined
- Attributes
  - Action type
  - Size – The size of the action in bits, not including the prefix
  - Prefix – The prefix is appended to action value
  - Prefix size – The size of the prefix in bits
- Input/Output – (Depends if WITH_ID was set)
  - Action ID – The action ID can be used as action type and acceptable in any API that accept `Bcm_field_action_type`

Error code: The SDK returns error if (size+prefix_size != Original action size)
- For example, if `bcmFieldActionSnoop` size is 20 bit, size+prefix_size must equal 20.

If action needs value mapping, use the `bcm_field_action_value_map()` API to map the required value to the corresponding HW value before inserting it to the HW:
- Input parameters:
  - Stage – The stage the action is being used in
  - Action – The user-defined action to map the value for
  - `bcm_value` – Value to map buffer, (maximum size 128b)
- Output parameters:
  - `hw_value` – Mapped HW value (to be inserted to HW)

11.6.4 Application Reference

Example of using user-defined actions and qualifiers in DE Field Group.
- **Type:** CINT example
- **Path:** `$SDK/src/examples/dnx/field/cint_field_basic_de.c`

Example of using user-defined action for creating drop action of 1 bit.
- **Type**: CINT example
- **Path**: `$SDK/src/examples/dnx/field/cint_field_action_drop.c`

Example of using `BCM_FIELD_ACTION_INVALIDATE`:
- **Type**: CINT example
- **Path**: `$SDK/src/examples/dnx/field/cint_field_efes_invalidate.c`

11.7 FG Qualifier/Action Offset

The qualifiers configured for a certain FG are internally mapped to a key. Each qualifier is assigned a place inside the key. Qualifiers are arranged next to each other so that no overlap or fragmentation is created.

Actions reside in the payload of the FG. Their arrangement is similar to that of the qualifiers.

Qualifier action offset get is used in the following cases:

- Cascading – One FG has a qualifier that reads data from the other FG payload
- Direct Extraction (DE) FG – Actions are derived directly from the qualifiers (qualifier offset needs to be given)

FG action offset get is used for the following reasons:

- Saves time calculating the desired offset for the qualifier/action to cascade/derive action from
- Preventing code duplications and bug prevention, as using this API will preserve the qualifier/action correct offset even when the queried FG has its key/payload structure changed.

NOTE: For the following field group types, the payload is MSB aligned.

- TCAM lookup
- TCAM direct table
- EXEM

NOTE: The offset returned is relative to the key LSB.

11.7.1 SOC Properties

N/A

11.7.2 Configuration Flow

1. `bcm_field_group_qualifier_offset_get()`
   a. Specify FG ID
   b. Specify Qualifier Type
   c. Returns the offset relative to the FG's LSB
   d. This function does not work for FGs with double key
2. `bcm_field_group_action_offset_get()`
   a. Specify FG ID
   b. Specify Action Type

11.7.3 Application Reference

Examples of using qualifiers and actions offset get APIs when implementing cascading Field Groups.

- **Type:** CINT example
- **Path:** `$SDK/src/examples/dnx/field/cint_field_cascading.c`

11.8 Advanced TCAM API

For TCAM Field Groups, extra functionalities and services are available to fully manage the TCAM database. These functionalities allow controlling TCAM bank allocations, deallocations, and clustering.

NOTE: The APIs are relevant only for the internal TCAM stage, unless it is explicitly indicated that the APIs are also supported for the external stage.

11.8.1 TCAM Bank Allocation Mode

When adding a TCAM field group (whether it is TCAM lookup), the user must specify the TCAM bank allocation mode (BAM).

Each TCAM bank can be used by only one stage of the pipe, for example:

- If a specific TCAM bank is allocated by FGs from the FLP stage, it cannot be allocated by any FG in the PMF stages.
- If a specific TCAM bank is allocated by an FG from the iPMF1 stage, it cannot be allocated by any FG in the iPMF2 or kPMF3 stages.

Currently, three TCAM bank allocation modes exist (specified by the enum `bcm_field_tcam_bank_allocation_mode_t`):

1. TCAM bank allocation mode AUTO—Default (`bcmFieldTcamBankAllocationModeAuto`):

   The field group starts with no banks and allocates banks on demand (when a request for entry add fails to find available space).

   When a TCAM field group is associated with this allocation mode, there is no explicit way to request a specific bank to be allocated for the Field Group.

2. TCAM bank allocation mode SELECT (`bcmFieldTcamBankAllocationModeSelect`):

   For TCAM groups with this type of allocation mode, the user can supply an additional array at creation time for specific bank IDs on a specific core to be allocated for the field group. In this mode, banks are not automatically allocated on demand, instead, an error is returned if no space is left to add an entry for this field group.

3. TCAM bank allocation mode SELECT with location (`bcmFieldTcamBankAllocationModeSelectWithLocation`):

   This mode is the same as `bcmFieldTcamBankAllocationModeSelect` but with following modification.

   In addition to bank selecting, field groups under this mode must specify the exact location of each added entry (when using `bcm_field_entry_add()`). This is achieved by specifying in the *priority* of each added entry the exact location where to add the entry.

11.8.2 TCAM Bank Allocation Mode Code Sample

```
bcm_field_group_t fg_id;
bcm_field_group_info_t fg_info;
bcm_field_group_info_t_init(&fg_info);
/* Type should be TCAM */
fg_info.fg_type = bcmFieldGroupTypeTcam;
fg_info.stage = bcmFieldStageIngressPMF1;
/*
 * Bank allocation mode is Select:
 * - Allocate 1 bank with id=5 when creating the FG
 * - Bank allocation on demand disabled in this mode
 */
fg_info.tcam_info.bank_allocation_mode = bcmFieldTcamBankAllocationModeSelect;
fg_info.tcam_info.nof_tcam_banks = 1;
```

```
fg_info.tcam_info.tcam_bank_ids[0] = 5;
/* Note that this function call would fail as no qualifiers/actions have been specified */
bcm_field_group_add(unit, 0, &fg_info, &fg_id);
```

11.9 TCAM Bank Add

The `bcm_field_bank_add()` API enables TCAM lookups:

- TCAM field groups that use `bcmFieldTcamBankAllocationdModeSelect` and `bcmFieldTcamBankAllocationdModeSelectWithLocation`
- PP applications that use TCAM for lookups (`bcm_field_app_db_t`)

Item	Auto	Select	SelectWithLocation
Number of banks and Bank ID	N/A	Relevant	Relevant
Bank added	Auto, when a new entry is added	`bcm_field_tcam_bank_add()`	`bcm_field_tcam_bank_add()`
Bank removed	Auto, when all entries are removed from the bank	`bcm_field_tcam_bank_evacuate()`	`bcm_field_tcam_bank_evacuate()`
New entry, location	Allocated automatically based on priority	Allocated automatically based on priority	Priority indicates the exact location of the entry
Allocation mode change after FG is added	Auto => Select, when FG has no entries and `bcm_field_tcam_bank_add()` is invoked (useful for PP TCAM tables)	Not possible	Not Possible

11.10 TCAM Bank Status

To retrieve the status of a certain TCAM bank, call the `bcm_field_tcam_bank_status_get` API. The TCAM bank status includes field groups that allocate the bank, the number of entries each has on the bank, and the number of total available entries per size on the bank.

For more information about this API, see Section 11.28, API Descriptions.

11.11 TCAM Bank Evacuation

This function allows evacuating a certain Field Group from a given set of TCAM banks. The evacuation process involves moving all the entries of the Field Group in those banks to other banks allocated by the Field Group to be evacuated. After the evacuation process is complete, the Field Group is removed from the banks and they are no longer allocated to the Field Group. If the Field Group has no entries on a given bank to be evacuated, the bank is simply removed from the Field Group allocated banks list.

TCAM Evacuation works only for TCAM Lookup Field Groups.

If not enough empty space is available in the rest of the banks allocated by the Field Group, the function fails after moving as many entries as possible from the requested banks. If the function manages to evacuate all the Field Group entries located on a certain bank, the Field Group is evacuated from the bank and the bank is removed from the Field Group allocated banks list.

11.11.1 TCAM Bank Evacuation API

`bcm_field_tcam_bank_evacuate(unit,flags,fg_id,&evac_info);`

- `flags` – Reserved for future use
- `fg_id` – Field Group ID to evacuate from the given banks
- `evac_info.nof_banks` – Number of banks to evacuate
- `evac_info.tcam_bank_ids` – Array of the banks evacuate
- `evac_info.core_ids` – Array of core ID for each bank

11.11.2 TCAM Bank Evacuation Code Sample

```
bcm_field_group_t fg_id;
/* Create FG with ID 'fg_id', and allocate it on banks 5 / 6 / 7 */
…
/*
 * Deallocate FG from bank 6 / 7 to allocate them for another FG in the same context.
 * Note: bank 5 has enough space to accommodate the FG entries to be removed from
 * bank 6 / 7.
 * Note2: For Double key FGs, only the even indexed bank is relevant.
 */
bcm_field_tcam_bank_evacuate_info_t evac_info;
evac_info.nof_banks = 2;
evac_info.tcam_bank_ids[0] = 6;
evac_info.tcam_bank_ids[1] = 7;
bcm_field_tcam_bank_evacuate(unit, fg_id, &evac_info);
```

11.12 TCAM Hit Indication

A special TCAM memory implements a sticky HIT indication for every entry in the TCAM memory space per core. If a passing packet triggers a TCAM lookup, the memory gets updated per TCAM bank with the value one for the bit that resembles the first entry that should hit in the given bank (for example, if a certain FG has entries on bank four and bank five and a passing packet triggered this FG lookup hitting an entry in bank four, with presumably another entry in bank five that would have hit otherwise, then both entries are marked as HIT in the hit indication memory. As if the hit indication memory simultaneously performs lookups on all given banks of the FG, and marks the entries that would otherwise hit if no entry in a higher priority bank existed).

The hit indication is sticky (persistent). As a result, dumping the memory after running multiple packets can often be misleading. Therefore, the `bcm_field_entry_hit_flush` function was introduced, with the FLUSH_ALL option in order to flush the whole memory and clear all sticking hit bit indications. Alternatively, use the function with a single entry handle as an input to clear only the hit bit of the entry pointed by the entry handle.

The `bcm_field_entry_hit_get` function returns whether hit indication is one for a given entry (hit indication='1' does *not* necessarily mean that the entry was hit).

NOTE:

- Hit indication APIs are only available for the field stage (although memory is also affected by non-field stages that use TCAM, there is no impact of any kind on the hit Indication APIs, since there is no bank sharing between different stages).

11.12.1 TCAM Hit Indication API

- `bcm_field_entry_hit_flush(unit,flags,entry_handle);`
  - `flags` – `BCM_FIELD_ENTRY_HIT_FLUSH_ALL` to clear all entry hit bits
  - `entry_handle` – Pointer to TCAM entry to clear its hit bit (can be NULL in case of FLUSH_ALL)
- `bcm_field_entry_hit_get(unit,flags,entry_handle,entry_hit_core_bmp);`
  - `flags` – Reserved for future use
  - `entry_handle` – A pointer to the TCAM entry to get ahit indication for
  - `entry_hit_core_bmp` – 2b output of hit indication per core

11.12.2 TCAM Hit Indication Sample Code

```
bcm_field_entry_t entry_handle;
uint8 entry_hit_core_bmp;
/* Create FG on bank 5 with one entry, save entry ID in entry_handle*/
…
/* Flush memory to check on clean state */
bcm_field_entry_hit_flush(unit, BCM_FIELD_ENTRY_HIT_FLUSH_ALL, NULL);
/* Send packet and see if entry was hit */
send_packet();

bcm_field_entry_hit_get(unit, entry_handle, &entry_hit_core_bmp);
if (entry_hit_core_bmp != 0)
{
    /* hit occurred on one of the cores */
}
```

11.13 TCAM Entry Cache

For each TCAM field group on any PMF stage, you can enable cache mode for adding entries. Cache mode can be enabled at any step after adding the field group. To add entries using cache, invoke cache mode. After that, each entry is added to the cache. After these steps, you can install all entries into the hardware or clear the cache.

NOTE: This feature is supported for the external TCAM as well as the internal TCAM.

11.13.1 Cache Configuration

Use the following steps to configure the cache:

1. Create the field group by calling `bcm_field_group_add()`.
2. Invoke the cache mode by calling `bcm_field_group_cache()` with cache mode `bcmFieldGroupCacheModeStart`.
3. Add entries to the TCAM entry cache.

   Call `bcm_field_group_entry_add()` to add entries. Each entry is saved in the cache and is not installed on the hardware until `bcmFieldGroupCacheModeInstall` is invoked.
4. Install all pending entries to hardware by calling `bcm_field_group_cache()` with cache mode bcmFieldGroupCacheModeInstall.

5. If deleting entries from cache is required, call `bcm_field_group_cache()` with cache mode `bcmFieldGroupCacheModeClear` to delete all pending entries, or call `bcm_field_entry_delete()` to delete specific entries.

NOTE:

- When called with `BCM_FIELD_FLAG_UPDATE`, the `bcm_field_entry_delete()`, `bcm_field_entry_info_get()`, and `bcm_field_entry_add()` APIs do not depend on cache mode. That is, they behave the same whether cache mode is on or off and whether the entry is within the cache or HW.
- The `bcmFieldGroupCacheModeClear` and `bcmFieldGroupCacheModeInstall` modes disable the cache mode. That is, `bcmFieldGroupCacheModeStart` must be re-invoked to enable cache.
- Cache configuration does not support warm boot.

11.13.2 Application Reference

Example of using TCAM entry cache:

- **Type:** CINT example
- **Path:** `$SDK/src/examples/dnx/field/cint_field_tcam_entry_cache.c`
- **Function:** `cint_field_tcam_entry_cache_main()`

11.14 Direct Extraction

Direct Extraction Field Groups (DE FGs) generate actions directly from a key. No lookups are performed.

A set of actions can be performed. Each action value can be composed of the following components:

- Specific bits from the key
- Constant values
- A combination of specific bits from the key and constant values.

DE FGs can be used in the iPMF2 and iPMF3 stages. Direct extraction is available in the egress stage.

NOTE: The BCM88690 does not support direct extraction in the egress stage.

Examples of using DE FGs:

- DestPort-ID = ITMH.DestPort
- Counter ID = OutLIF-ID
- If (1000 < src_port < 2000): if (TTL < 64) double TTL; otherwise, set it to 64
- If (200 < vlanId < 800), increase TTL by 5; otherwise, set it to 127

11.14.1 SOC Properties

N/A

11.14.2 Configuration Flow

To create and use a Direct Extraction Lookup:

1. Add a field group with qualifiers and actions, by calling `bcm_field_group_add` API with `FG_Type` = Direct Extraction.
   The qualifiers define the FG's key structure. Key structure is {Q*n*, …, Q1, Q0}:
   – Qualifier 0 is added to LSB Following qualifiers follow in the direction of MSB…
   – Qualifier (n-1) is the MSB of the key
   – Tip: Use user-defined qualifiers to only select subset of the qualifier's bit
   The actions extract the action values from the key. Action structure is {A*m*, …, A1, A0}
   – The value of Action 0 is the LSB of the key
   – Following actions are extracted from the next portions of the FG key in the direction of the MSB
   – The value of Action (*m* – 1) is the MSB of the key
   Use the action bcmFieldActionVoid when the key bits are only used for FEMs

NOTE: The total sizes of {Qn, …, Q1, Q0} and {A*m*, …, A1, A0} must be equal. When action in the FG is set `with_valid_bit`, an additional qualifier bit is expected.

2. Optional: Use FEM APIs to add actions that require FEM capabilities (for a detailed explanation, see Section 11.16, Field Extraction Macro.

3. Attach the field group to context by calling `bcm_field_group_context_attach()`: Attach the field group to contexts according to attachment attributes
   Same behavior as for TCAM Field Groups:
   – Key: FG ID, Context
   – Action info.
   • Per action data: Priority: The priority of the action
   – Qual Info:
   • Select the qualifiers source

NOTE:

- There is no need to add entries for Direct Extraction Field Group.
- Typically, Direct Extraction Keys share the key with the TCAM Field group to use the same Key for 80b TCAM field group and Direct Extraction. The TCAM field group must be attached first to the context. The reverse order will fail. This is true for iPMF2 and iPMF3.

11.14.3 Application Reference

Using Direct Extraction Field-groups.

- **Type:** CINT example
- **Path:**

```
$SDK/src/examples/dnx/field/cint_field_action_prefix.c
$SDK/src/examples/dnx/field/cint_field_input_type_const.c
$SDK/src/examples/dnx/field/cint_field_de_conditions.c
$SDK/src/examples/dnx/field/cint_field_dir_ext_epmf.c
```

11.15 Cascading

Cascading uses the result of a specific stage for classification or context selection in a later stage.

11.15.1 Result Cascading

Cascading uses results from one FG as qualifiers for another FG. The second FG is used for lookups in later stages.

There are two types of cascading results:

- **Standard**: A defined place in the cascaded payload. For example, FG in iPMF2 uses a TC result from iPMF1 FG.
- **Void:** The location of the data to use needs to be defined explicitly. For example, FG in iPMF2 use bits 32:10 from iPMF1 FG result buffer.

Possible cascading options:

- iPMF1 to iPMF2
- iPMF1/2 to iPMF3
- iPMF1/2/3 to ePMF

11.15.1.1 Using Standard Results

For example, FG in iPMF2 uses a TC result from iPMF1 FG:

- `bcm_field_group_add()` and `bcm_field_entry_add()`:
  Usage is not affected by cascading (In fact, the same iPMF2 FG with TC qualifier can have the TC):
  - Extracted from the Metadata in context 1
  - Extracted from iPMF1 FG in context 2 (Cascading)
- `bcm_field_group_context_attach()`
  - Qual ID: TC (User-defined qualifier called TC)
  - Input Type: Cascading
  - Input Arg: FG ID. (The FG ID from which the action should be extracted)
  - Offset – Use `bcm_field_group_action_offset_get()` to get the offset in the buffer of the cascaded from Field group

11.15.1.2 Using Void Actions

A user defines the results in one FG that will later be used as a qualifier in another FG. For example, when iPMF1 maps traffic to profiles and iPMF2 uses the profile as a qualifier.

When iPMF2 FG uses void results from iPMF1 FG:

- iPMF1 FG will get `Action_id` created by `bcm_field_action_create()` as one of its actions
- iPMF2 FG will get `Qual_id` created by `bcm_field_qualifier_create()` as one of its qualifiers
- When attaching context to the iPMF2 FG, the Qual ID's offset will point to the `Action_id` location.

11.15.2 iPMF1/2 to iPMF3 Cascading

11.15.2.1 Standard Results

Between iPMF2 and iPMF3, the Pipeline Metadata is updated by iPMF2 actions:

- Therefore, cascading qualifier input is not required.
- For example, to use the destination result from iPMF2, use input type = metadata and not input type = cascading.

11.15.2.2 Void Results

On top of the standard actions, there are result buffers for iPMF2 → iPMF3 cascading:

- General data
  - `bcmFieldActionContainer` – Action Container uses the *General data* buffer, which might be used/filled by stages previous to iPMF1/2, (it is permissible to override it in iPMF). If iPMF3 uses QualifyContainer, then iPMF1/2 must reset the action to 0 to avoid a false positive hit in iPMF3 if it was not filled with real data in iPMF1/2.
    - As solution ConstField group with low priority EFES can be set with value 0. Any lookup that must be set to ActionContainer should use EFES with a higher priority.
- UDH words
  - `bcmFieldActionUDHData`... – If `bcmFieldUDHBase`… is not 0, the data will be sent to egress. See Section 11.15.3, Ingress to Egress Cascading (UDH).

11.15.3 Ingress to Egress Cascading (UDH)

iPMF1/2/3 has dedicated actions to configure UDH buffer size and fill it with the required data:

- UDH (User-defined Header) is composed of
  - 8 bit base header. Each 2 bits encode the size and type for each of the UDH data buffers
    - 0 – Mapped to UDH Size: 0b, Type: None
    - 1 – Mapped to UDH Size: 32b, Type: PMF cascading

      In the BCM88690 BX device, value 1 of UDH0 is for IFA 1.0. For more information, see Chapter 50, Instrumentation – Inband Flow Analyzer

      ePMF context selection can ignore such packets by qualifying on `bcmFieldQualifyFtmhAsePresent=1` and `bcmFieldQualifyFtmhAseType = INT`.

      UDH 1 to 3 can still be used for cascading purposes.
    - 2 – Mapped to UDH Size: 32b, Type: EVPN application
    - 3 – Mapped to UDH Size: 32b, Type: SRv6 application
  - 4 UDH data buffers. Each buffer is 0/32 bits long (the size of each buffer is determined by the UDH base header)
- The `bcmFieldActionUDHBase` action is used to set the sizes of the device use of four UDH data buffers
- `bcmFieldActionUDHData` 0-3 actions are used to fill in each of the four UDH data buffers
- `bcmFieldQualifyUDHBase` is used to read the data from the UDH base header
- `bcmFieldQualifyUDHDatav` 0-3 qualifiers are used to read the data from each of the four UDH data buffers

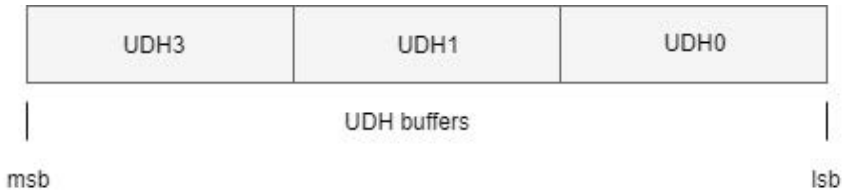
UDH Life Cycle

- UDH buffers are built in Ingress side, according to the sizes listed in the base

Figure 5: UDH Base/Data Buffers Structure

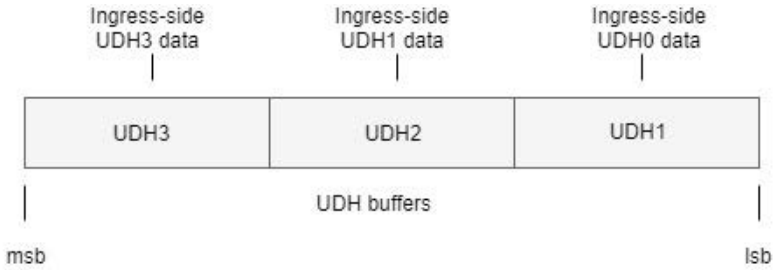
- The buffers are built in ascending order starting the buffer of UDH0 in lsb
- Example for (UDH0=4B, UDH1=4B, UDH2=0B, UDH3=4B):

Figure 6: Example of the UDH Data Buffers Structure in the Ingress



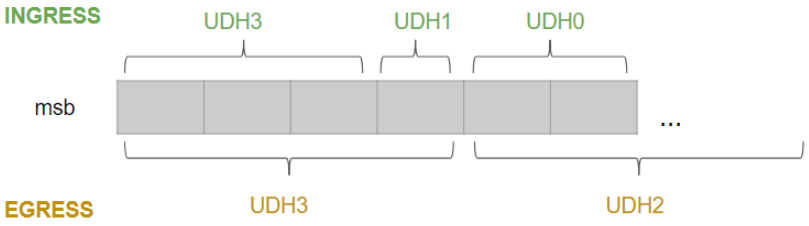
- When the buffers are transferred to the egress, the buffers are copied as MSB aligned and a new buffer is created. In contradiction to the ingress, the egress buffers size is static, and is 4B per each UDH buffer. As copying is MSB aligned, the first buffer to get filled is UDH3, and in case the UDH buffers in the ingress are less or equal to 4B in size, only the UDH3 buffer is valid in the egress, regardless of which buffers were filled in the ingress.
- Example for the previous buffer (12B in size) after being transferred to the egress side:

Figure 7: Example of UDH Data Buffers Structure in the Egress



- To conclude, when setting certain UDH data buffers in the ingress, they do not necessarily map to the same UDH data buffers in the egress.

  The following figure (which assumes dynamic UDH sizes) summarizes the differences between the UDH buffers in the ingress and egress sides.



NOTE:

- UDH_Data in the egress can be junk, so it is important to validate the relevant UDH_Base bits were set to perform lookup on the UDH_Data.
  The UDH_Base can be either part of the context selection or part of the key for the lookup.
- Use UDH buffers for iPMF1/2 to iPMF3 cascading as well.
- As a best-practice, it is advised to use the UDH data buffers in descending order, so that it best fits the qualifiers in the egress, especially when the buffers are 4B in size (that is, when two buffers of size 4B are needed, using UDH2-3 in the ingress is mapped to UDH2-3 in the egress in the right order, enabling the same number treatment of the buffer in both sides).
- UDH can be used for other means besides the PMF cascading.

- iPMF1 to iPMF2 cascading for 320for double-key sized FGs does not support cascading from actions that span the first and second halves of the payload (that is, include bits 63-64).
  One workaround for this limitation is to use two cascading qualifiers for each part of the split action. Another way is to change the actions order so that the actions that are to be cascaded from are not split.
- UDH_Type value 0x55 is a predefined value used in the SDK to indicate an SRv6 endpoint flow without decreasing SRv6.SL.

For an example, see Section 11.15.6, Configuration Flow.

11.15.4 Context Update

Context selection according to data from previous FGs. For example, if (FG-3.TC == 3 & iPMF1 context_profile == 6): iPMF2 context = 9.

In this example, FG-3.TC is implemented by:

- Using the `bcmFieldQualifyCascadedKeyValue` qualifier, which uses the 4MS bits of the TCAM result which FG3 is using and constructing the payload buffer of this FG such as TC will reside on those bits.

The `cascasded_from` attribute of iPMF2 the context should be set to the iPMF1 context-ID.

There is no reason to switch context to iPMF2 unless additional qualifiers in CS are used (otherwise use iPMF1 context configuration).

When the presel set API is called for iPMF2 context with additional qualifiers, the SDK sets the context profile qualifier which was saved previously.

If there is no hit in iPMF2 context selection table, iPMF2 context ID remains as iPMF1 context ID.

This option only relevant between iPMF1 and iPMF2 stages.

The only field groups in iPMF1 that can be *cascaded from* for a context update are TCAM field groups. Furthermore, only full-key TCAM field groups are supported.

11.15.5 SOC Properties

N/A

11.15.6 Configuration Flow

When iPMF2 FG use flexible results from iPMF1 FG Call:

- `bcm_field_action_create("name", size, &action_id)`
- `bcm_field_qualifier_create("name", size, &qual_id)`
- `bcm_field_group_add(fg_1, stage=iPMF1, Qual={...}, Action={...,action_id})`
- `bcm_field_group_add(fg_2, stage=iPMF2, Qual={...,qual_id}, Action={...})`
- `bcm_field_group_context_attach(fg_1, Stage: iPMF1, ...)`
- `bcm_field_group_action_offset_get(fg_1, action_id, &action_offset)`

- `bcm_field_group_context_attach()`
  - `FG ID=fg_2`
  - `Stage= iPMF2`
  - `Qual ID[X]= qual_id`
  - `Qual ID[X].Input Type= Cascading`
  - `Qual ID[X].Input Arg= fg_1`
  - `Qual ID[X].Input offset=action_offset`

11.15.7 Application Reference

Using cascading in Key construction between iPMF1/2 and iPMF3

Example of cascading Field Groups.

- **Type:** CINT example
- **Path:** `$SDK/src/examples/dnx/field/cint_field_ipmf2_ipmf3_cascading.c`

Context update from iPMF1 to iPMF2

Example of context update between iPMF1 and iPMF2

- **Type**: CINT example
- **Path**: `$SDK/src/examples/dnx/field/cint_field_contexts_cascading.c`

11.16 Field Extraction Macro

Field Extraction Macro (FEM) is a HW machine that is capable of invoking a specific action (refer to Section 11.2, Field Groups) out of a set of actions, depending on some conditions. FEM HW offers 16 FEM machines.

FEMs are executed according to priority starting from the lowest index FEM up to the highest index FEM (as is the case with EFESs). If two FEMs perform the same action, the FEM with the higher index takes precedence.

Multiple Configurations for Different FGs

The same FEM machine (FEM ID) can be reused up to four times by different FGs with totally different configurations.

No two of these FGs can be attached in the same context.

NOTE:

- An FG cannot reuse the same FEM ID.
- Unlike FES HW, which can be found after nearly every PMF stage to invoke actions with, FEM HW has only one instance after the iPMF-2 stage. Therefore, only Field Groups configured on the iPMF-1 and iPMF-2 stages can use FEMs.
- FEMs are supported only for TCAM, EXEM, and direct extraction field groups.

A limitation is that FEMs cannot be used on the 12 LSBs of the maximum payload size of exact match field groups.

11.16.1 FEM Configuration

11.16.1.1 FEM Input Selection

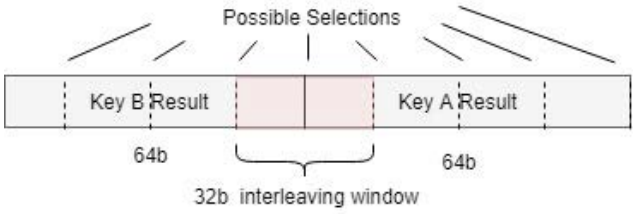
Each FEM uses up to 32b payload. The payload for each FEM can be selected from various sources including:

- TCAM lookup payloads
- Direct Extraction FGs key (iPMF-2 stage only)

NOTE:

- It is best practice to use the `bcm_field_group_actions` or `quals_offset_get()` APIs to determine the in-key offset for the FEM.
- The FG result can be of any action (not necessarily void), and the selected FEM input can span multiple actions payload.
- The FEM input is a 32b payload window extracted from the FG result and can be chosen with a 16b granularity. Because the results are continuous in the memory for double-key FGs, a 32b interleaving payload that extracts 16b from each key result can be selected.

Figure 8: Sample Double-key 32b Interleaving Window



11.16.1.2 Action Selection

Up to four actions and up to 16 conditions based on four condition bits can be configured for each FEM:

The four condition bits can be any four consecutive bits in the 32b input payload. Configure the `condition_msb`, which is the MSB of the four condition bits (the minimum value is three).

For each condition (out of the 16 conditions based on the four condition bits selected). Specify whether to run an action, and which out of the four (configured) actions to run.

11.16.1.3 FEM Action Configuration

FEM actions are unique, highly flexible and enable controlling any of the input bits of the action. For each FEM machine up to four actions can be configured, the FEM action is composed of two values (action type and action adder) and one array (field select).

- Action type – Action to perform, should be one of supported action types that can be found in Table 23, iPMF-1,2 Actions. Alternatively, run the field actions predefined BCM command on the device, to see a list of the supported actions.
- Field select (array) – This array of size 24 describes how to compose the 24b payload of the action, for each bit, pick either forced value of either 0 or 1, or the value in the input key. Actions that require more than 24 bits are not permitted.
- Action adder – The value to add to *action value* before invoking the action that is specified by *action type*.

NOTE: FEM-IDs 0 and 1 can only support actions of up to four bits. They do not include support for the action adder fields.

11.16.1.4 Invalidating FEMs by Context

The FEM `add` API is given per field group, so it is applicable to all contexts attached to that field group. Sometimes having a different FEM configuration for different contexts is required, or else two field groups attached to the same contexts have the same FEM position.

To solve this type of case, an option is available to invalidate a certain FEM during group context attachment by providing its position in the array (`payload_info.invalidate_fems`, *`number of elements in array`* `payload_info.nof_invalidate_fems`).

NOTE: The `payload_info.invalidate_fems` field is a pointer initialized to NULL, and the caller is expected to provide a pointer to a memory where the array would be located.

11.16.2 SOC Properties

N/A

11.16.3 Operation and Configuration

In standard operation, follow the following:

- FEMs are added one by one, to a Field Group (FG) which has been created (`bcm_field_group_add()`) before any context has been attached to that FG. Refer to Section 11.2, Field Groups.
- The same FG can not add (`bcm_field_fem_action_add()`) the same FEM again. It must first be removed (`bcm_field_fem_action_delete()`).
- FEMs may only be removed (`bcm_field_fem_action_delete()`) after all contexts have been detached (`bcm_field_group_context_detach()`).

11.16.4 Configuration Flow

1. To add a FEM to a Field Group (FG), the Field Group must first be created:

   Call `bcm_field_group_add()`.

   Actions to be invoked by the FG can be specified. To have data in the result of the FG for the FEM to operate on that would otherwise not exist, void actions are available.

2. FEM is added to the FG. To add a FEM, the caller should have a clear idea regarding the following:
   - FEM key select in general and specifically the location of the 32-bits chunk within it.
   - The location, within the 32-bits chunk, of the 4-bits which are to serve as *condition* and the various actions (up to four) which are to be activated by each condition.
   - How to construct the (up to 24b) value of the action, bit by bit (either from the 32-bits chunk or from hard coded values).

   Call `bcm_field_fem_action_add(unit, flags, fg_id, action_priority, &fem_action_info);`

   The input is:
   - `fg_id` – The Field Group to add FEM to.
   - `action_priority` – Encoded position of the FEM to add to this FG. Essentially, this is the identifier of the FEM. Caller calculates this value using `BCM_FIELD_ACTION_POSITION` with `array_id` set to 1 or 3 and *position* set to 0 to 7.

  - `fem_action_info` – Pointer to structure:
    - `fem_input`
      - `uint8 input_offset` – The offset, of the input result, of the 32-bits chunk.
      - `bcm_field_group_t overriding_fg_id` – The Field Group that supplies 16 LB bits to replace. If invalid, no such operation is required.
    - `condition_msb` – The MS bit, on the 32-bits chunk of the four bits which are to be used as a condition.
    - `fem_condition` – An array of 16 structures, one per 'condition' which indicates, for each:
    - `uint8 extraction_id` – Identifier of the action descriptor to use (one of four which is detailed in `fem_extraction` below)
    - `uint8 is_action_valid` – Boolean flag. If zero, no action is carried out on this condition.
  - `fem_extraction` – Array of four action descriptors:
    - `action_type` = Action to carry out.
    - `output_bit` – Array of 24 bit descriptor indication on how to construct action value. For each bit, indicate whether to take it from the 32-bits chunk (`bcmFieldFemExtractionOutputSourceTypeKey`) or from a value written on this input (`bcmFieldFemExtractionOutputSourceTypeForce`). In the first case, an offset within the chunk is specified. In the second, `forced_value` (0 or 1) is specified. For FEM 0/1 there are only four bits.
    - `increment` – Value to add to calculated action value. Mot available for FEM 0/1.

3. To delete the FEM from the Field Group, detach all contexts and call `bcm_field_fem_action_delete(unit, fg_id, action_priority)` with the following parameters:
   - `fg_id` – The Field Group to remove FEM from
   - `action_priority` – Encoded position of the FEM to remove from this FG. Essentially, this is the identifier of the FEM. Calculate this value using `BCM_FIELD_ACTION_POSITION` with `array_id` set to 1 or 3. Set position to 0 to 7.

11.16.5 Application Reference

FEM Initialization

Example of general FEM initialization, 0x9100, 0x88a8.

- **Type:** General setup cint, including increment (adder)
- **Path:** `$SDK/src/examples/dnx/field/cint_field_fem_increment.c`
- **Function:** `field_fem_increment_index_config_ipmf2_de()`

FEM Application

- **Type:** CINT example
- **Path:** `$SDK/src/examples/dnx/field/cint_field_fem.c`
- **Function:** `cint_field_fem_main()`

FEM Invalidate by Context

- **Type**: CINT example
- **Path**: `$SDK/src/examples/dnx/field/cint_field_fem_invalidate.c`
- **Function**: `cint_field_fem_invalidate_main()`

11.17 Enhanced Field Extraction Shifter

The Enhanced Field Extraction Shifter (EFES) is a HW machine that is capable of invoking a specific action (see Section 11.4, Qualifiers and Section 11.5, Actions) out of a set of actions, depending on some conditions, similar to FEM.

EFESs are executed according to priority starting from the lowest index up to the highest index (as is the case with FEMS). If two EFESs perform the same action, the EFES with the higher index wins.

Multiple Configurations for Different FGs

The same EFES machine (EFES ID) can be reused up to 31 times by different FGs that have totally different configurations, with the sole condition that no two of these FGs can be attached in the same context.

NOTE: The same FG cannot reuse the same EFES ID.

Capabilities of the EFES Machine

An EFES has the following capabilities:

- It can extract up to 32 bits from the field group result and write them to an action. (For direct extraction field groups, the result is the same as the key.)
- It can use one valid bit (reading the bit before the LSB of the action to decide whether to perform the action).
- It can apply a bitwise OR mask to the value taken from the field group result.
- It can use four different configurations depending on two condition bits. However, not all bits on the FG result can be used as condition bits.

Ways to Configure an EFES

Whenever a field group is created and attached to a context, for each nonvoid action, an EFES machine is automatically configured (unless invalidated by priority). When used in this way, the bitwise OR mask implements the action prefix, and all four configurations are identical, so no EFES 2-bit condition is used.

A second way to configure an EFES is by using the function `bcm_field_efes_action_add()`. Unlike `bcm_field_fem_action_add()`, EFESs are added after the field group is attached to a context. This helps address the following scenarios:

- Using different actions types for the same lookup database for different contexts.
- Using the same bits from the lookup result for more than one action.
- Using different actions for different entries (with the condition feature)
- Using a 3-bit condition in direct extraction (using the two condition bits plus a valid bit with polarity).

When using `bcm_field_efes_action_add()`, the priority of the actions must be managed by position.

NOTE:

- When using `bcm_field_efes_action_add()` with a condition on a direct extraction field group, the `BCM_FIELD_FLAG_32_RESULT_MSB_ALIGN` flag must be used during context attach. If used together with FEMs, there are further limitations on the field group's result size.
- The added EFES can be used together with actions provided when creating the field group. Alternatively, it is possible to make all actions void actions when creating the field group and have all actions added later.

11.17.1 Added EFES Configuration

11.17.1.1 EFES Condition

The EFES can be added with or without a condition. If no condition is used, all four configurations will be configured the same, and there are no restrictions on where the data is taken from.

When using a condition, the location from where the data is read has a restriction. The two condition bits are bits 62–63 of the action result buffer, and every 2 bits 32 bits thereafter (bits 94–95, bits 126–127, and so on). That means the field group using the first key in each stage (not including iPMF2) would have fewer possible locations for condition bits than field groups using other keys (that is, the condition bits cannot be placed on bits 30–31 of the field group's result buffer).

For most instances, the 2 bits of the last action in the field group are legitimate condition bits and every 32 bits before them.

NOTE: To ensure proper alignment with this rule, EFESs with a condition can be added to a direct extraction field group only if it was attached to the context with the flag `BCM_FIELD_FLAG_32_RESULT_MSB_ALIGN`. If the field group also used FEMs, it must be at least 32 bits in size, and its size must be a multiple of 16.

Another constraint is that all of the data taken from the field group's result (including the valid bit, if it exists) must come from the 60 bits prior to the two condition bits or the two condition bits themselves.

For example, when using a double-key TCAM field group with 80 bits of payload, it is possible to use the following bits:

- Bits 78–79 as condition with input starting from bit 16 or above.
- Bits 46–47 as condition with input starting from bit 0 or above.
- Bits 14–16 as condition with input starting from bit 0 or above.

If the condition bits given are illegal, the error message provides all legal values for condition bits on the field group-context pair.

NOTE:

- It is a better practice to use the `bcm_field_group_actions` or `quals_offset_get()` API to determine the in-key offset for the EFES.
- The FG result can be of any action (not necessarily void), and the selected EFES input can span the payload of multiple actions.

11.17.1.2 EFES Configuration

Consider an array with four elements. Using a condition requires filling all four elements; otherwise, only the first must be filled.

For each option, it is possible to determine that no action will take place or to choose a specific action. Using a valid bit means that the action to be performed only takes place if the value of the valid bit (the bit before the first bit taken as input) equals the value of the valid bit's polarity.

It is also possible to apply a bitwise OR mask to the value of the action. However, the masks are a more limited resource than EFES configurations, and any mask used other than 0 consumes that resource.

11.17.2 Configuration Flow

11.17.2.1 Different Action Types for Different Contexts

1. Create the field group.

   Call `bcm_field_group_add()`.

   Specify any actions to be invoked by the FG. To generate data (that would otherwise not exist) in the FG's result for the added EFES to operate on, void actions are available. This configuration example uses one action (void or otherwise) known as *action A*.

2. Call `bcm_field_group_action_offset_get()` to get the offset of action A.

3. Attach the field group to multiple contexts.

   Call `bcm_field_group_context_attach()` for each context.

4. Add an EFES for each context.

   Call `bcm_field_efes_action_add(unit, flags, fg_id, context_id, action_priority, &fem_action_info)`.

   – Initialize `fes_action_info` with `bcm_field_efes_action_info_t_init()`.
   – Set `is_condition_valid` to false.
   – Only set the first element of the array `efes_condition_conf`.
     - Set `is_action_valid` to true.
     - Set `action_type` to the specific predefined action type required for the context.
     - Set `action_lsb` to be the offset of action A (or +1 if action A is a void action but includes a valid bit).
     - Set `action_size` to be the size of action A (or –1 if action A is a void action but includes a valid bit).
   – `action_priority` is the location of the EFES (use `BCM_FIELD_ACTION_POSITION` macro).

11.17.2.2 Multiple Actions Using the Same Bits

1. Create the field group.

   Call `bcm_field_group_add()`.

   Have the field group include an action knwon as *action B*.

2. Call `bcm_field_group_action_offset_get()` to get the offset of action B.

3. Attach the field group to the context

   Call `bcm_field_group_context_attach()`.

4. Add an EFES for each extra action.

   Call `bcm_field_efes_action_add(unit, flags, fg_id, context_id, action_priority, &fem_action_info)`.

   – Initialize `fem_action_info` with `bcm_field_efes_action_info_t_init()`.
   – Set `is_condition_valid` to false.
   – Only set the first element of the array `efes_condition_conf`.
   - Set `is_action_valid` to true.
   - Set `action_type` to the specific predefined action type that we want to add.
   - Set `action_lsb` to be the offset of action B
   - Set `action_size` to be the size of action B (can be retrieved using `bcm_field_action_info_get()`).
   - Set `action_with_valid_bit` to be the same as action B.
   - Set `valid_bit_polarity` to be the same as action B.
   – `action_priority` is the location of the EFES (use `BCM_FIELD_ACTION_POSITION` macro).

11.17.2.3 Using Different Actions for Different Entries

1. Create the field group.

   Call `bcm_field_group_add()`.

   Have the field group include an action known as *action C*, a void action with two bits as the last action.

2. Call `bcm_field_group_action_offset_get()` to get the offset of action C.

3. Attach the field group to context.

   Call `bcm_field_group_context_attach()`.

4. Add an EFES for each action.

   Call `bcm_field_efes_action_add(unit, flags, fg_id, context_id, action_priority, &fem_action_info)`

   – Initialize `fem_action_info` with `bcm_field_efes_action_info_t_init()`.
   – Set `is_condition_valid` to true.
   – Set `condition_msb` to be the offset of action C +1.
   – Set for each element in `efes_condition_conf`, to be performed according to the corresponding value of action C.
   - Set `is_action_valid` to true or false, according to whether or not we want the action to be performed when action C has this value.
   - Set the rest of the structure according to what we want to be performed when the value of action C is equal to the element index.
   – `action_priority` is the location of the EFES (use `BCM_FIELD_ACTION_POSITION` macro).

11.17.2.4 Using More Than One Bit Condition for Direct Extraction

1. Create the field group.

   Call `bcm_field_group_add()`.

   Have the field group include an action known as *action D*, a void action with two bits as the last action.

2. Call `bcm_field_group_action_offset_get()` to get the offset of action D.

3. Attach the field group to context.

   Call `bcm_field_group_context_attach()`.

   Use the flag `BCM_FIELD_FLAG_32_RESULT_MSB_ALIGN` because a direct extraction and condition are used.

4. Add an EFES for each action.

   Call `bcm_field_efes_action_add(unit, flags, fg_id, context_id, action_priority, &fem_action_info)`.

   – Initialize `fem_action_info` with `bcm_field_efes_action_info_t_init()`.
   – Set `is_condition_valid` to true.
   – Set `condition_msb` to be the offset of action D +1.
   – Set for each element in `efes_condition_conf`, to be performed according to the corresponding value of action D.
   – `action_priority` is the location of the EFES (use `BCM_FIELD_ACTION_POSITION` macro).

11.17.3 SOC Properties

N/A

11.17.4 Application Reference

Using EFES add

- **Description**: Examples of using EFES add
- **Type:** CINT example
- **Path:** `$SDK/src/examples/dnx/field/cint_field_efes_add.c`
- **Functions**:
  - `cint_field_efes_add_diff_actions_diff_contexts_main()`
  - `cint_field_efes_add_two_actions_same_data_tcam_main()`
  - `cint_field_efes_add_two_actions_same_data_de_main()`
  - `cint_field_efes_add_diff_actions_diff_entries_main()`
  - `cint_field_efes_add_de_three_bit_condition_main()`

11.18 Hash

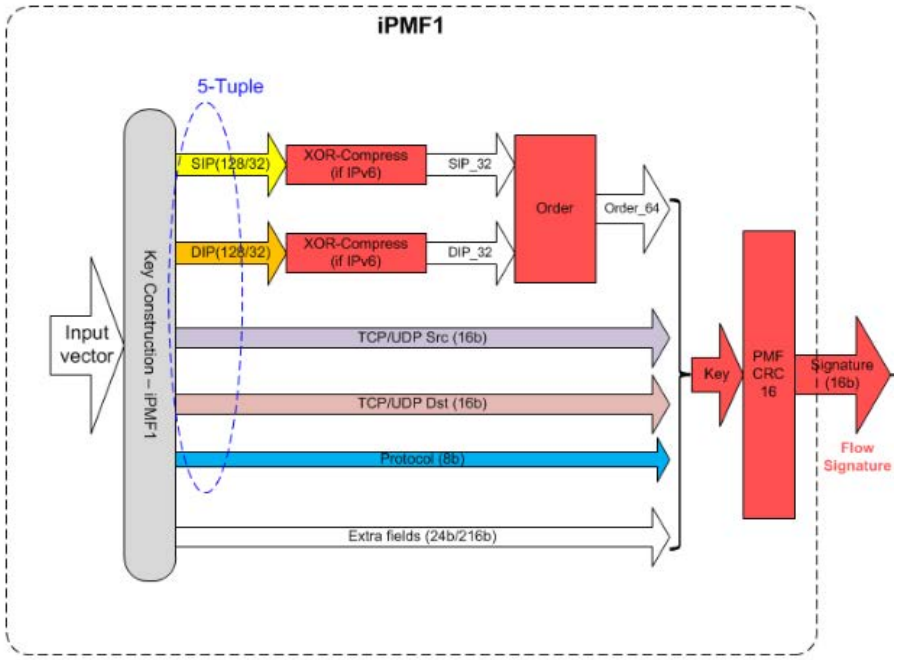
iPMF1 stage supports an operation of compression and hashing up to 320-bit key to several 16-bit outputs, which can be used as a qualifier for lookups in the iPMF2 stage.

The typical use of this functionality is done by some instrumentation applications (such as trajectory trace) includes: 5-tuple flow classification, compressing the information and hashing it to a 16-bit signature, which can be used for further processing and action resolution.

The following functionality is supported:

- Compression (for IPv6):
  - Compression is XOR 4-fold:
  - Collapse32_Field_128 = Field_128[127:96] ^ Field_128[95:64] ^ Field_128[63:32] ^ Field_128[31:0]
- Order
  - Symmetric_IP_Address_0 = Min(IP Source Address, IP Destination Address)
  - Symmetric_IP_Address_1 = Max(IP Source Address, IP Destination Address)
- Hash (PMF CRC-16)
  - The user chooses the hash function (should be the same for all nodes)
  - Can update LB keys

Figure 9: Typical Usage of Hashing Functionality with 5-tuple Classification



The PMF CRC-16 unit:

- Can perform the following Hash calculations on the extracted key. The maximum key size is 320b (entire initial keys i,j).
- The Configurable CRC unit supports several Hash functions, such as CRC16 Bisync, CRC32, XOR16, and so on. Configurable user per PMF-context. For additional details about the supported function, see Table 5, CRC-16 Functions Enum.
- Is combined with the other pipe CRC.

11.18.1 SOC Properties

N/A

11.18.2 Configuration Flow

To configure HASH key which is used to create CRC call:

- `bcm_field_context_hash_create()` – Create a new hash key on specific context

  Parameters:
- `flags` – uint32, to update the hash configuration of existing context, not including `key_info`. (`BCM_FIELD_FLAG_UPDATE`)
- `stage` – In BCM88690, only iPMF-1 is supported
- `context` – Context to configure the hash for (the context needs to be created with hash support for this function to work)
- `hash_info`:
  - `key_info`
    - `qual_type` – Type of `bcm_field_qualify_t`, to indicate which qualifiers build the key (Validate the total qual size is under 320)
    - `qual_info` – `attach_qualify_info`, same struct as used in `context_attach`
      - Input type
      - Input arg
      - Offset
  - `compression` – Disable or enable data compression (from 128 bits to 32 bits). Typical usage is to compress an IPv6 address into 32 bits.
  - `order` – Disable or enable the order of data (smaller 32-bit value will always be in 32LSB). Typical usage is to order between generating the same key for bidirectional IP flows.
  - `hash_function` – Hashing polynomial for hashing mechanism. For supported values, see Table 5, CRC-16 Functions Enum.
  - `hash_config` – Hashing configuration parameters, describing which hash action is performed and which signals.
    - `action_key` – Key to Config.
    - `function_select` – Hashing function to perform on the key. For supported values, see Table 6, Hash Function on Key.
    - `crc_select` – If the hashing function selected is one of the augmentation functions, `crc_select` selects the CRC functions to be performed on the augmentation output.
  - `nof_additional_hash_config` – Number of additional hash config data.
  - `additional_hash_config` – Array of additional `hash_config`. This array is a continuation to `hash_config`.

NOTE: Only `hash_function` and `hash_config` (including `additional_hash_config`) can be updated when calling `bcm_field_context_hash_create()` with `BCM_FIELD FLAG_UPDATE`.

- `bcm_field_group_context_attach()` – Attach FGs to the new context
- `bcm_field_presel_set()` – Map traffic to the context (the default context is chosen when there is no hit).

11.18.3 Application Reference

Using hash keys:

- **Type**: CINT example
- **Path**: `$SDK/src/examples/dnx/cint_field_hashing.c`
- **Function**: `cint_field_hashing_main()`
- **Path**: `$SDK/src/examples/dnx/cint_field_hash_ipt.c`
- **Function**: `cint_field_hash_ipt_main()`

Simulator to generate hash keys:

- **Type**: CINT example
- **Path**: `$SDK/src/examples/dnx/field/cint_field_crc_hash_sim.c`
- **Function**: `cint_field_crc_hash_sim_main()`

Using multiple action keys:

- **Type**: CINT example
- **Path**: `$SDK/src/examples/dnx/field/cint_field_6_hash_keys.c`
- **Function**: `cint_field_6_hash_keys_main()`

11.19 Compare

This feature compares two 32-bit values and use the result as a qualifier for a later lookup or context selection.

iPMF1 stage supports these operations:

- Comparing one or two pairs of 32-bit values stored in the 32 MSB of initial keys.
- Comparing one or two pairs of 32-bit values stored in one of the iPMF1 initial keys with a TCAM lookup result.

The keys to compare are built in iPMF1 using the same method as regular keys (using qualifiers).

The result of the comparison can be used by the iPMF2 stage for:

- Context selection.
- Key construction for a lookup.

Comparison operands: equal, not-equal, greater than, smaller-than, not bigger than, not smaller than. Example usages:

- IPv4_Sip > IPv4_Dip?
- OutLIF > 3000?
- InLIF != FG-X.result.OutLIF?

Context Compare Modes:

Upon creating the context the user must specify which compare keys to reserve using `context_compare_modes`. This is needed because the SDK allocates bits from the initial keys F-I according to mode selection: `compare_1_mode` sets the compare mode for the first pair. `compare_2_mode` sets the compare mode for the second pair.

The compare mode options (for each of the two pairs):

- `bcmFieldContextCompareTypeNone` (No compare mode)
- `bcmFieldContextCompareTypeSingle` (Use only the first key comparison against TCAM-result)
- `bcmFieldContextCompareTypeDouble` (Use both keys comparison along with comparing the first key against TCAM-result)

For an example of the usage, see Section 11.19.3, Application Reference.

Compare initial key with TCAM FG result:

- `bcm_field_group_add()`
- `bcm_field_group_context_attach()` has a dedicate info for compare `BCM_FIELD_CMP_1`/`BCM_FIELD_CMP_2` compares the 32 msb of the FG result with a context comparison keys
  - `BCM_FIELD_CMP_1` will compare the 32 msb of the TCAM result to comparison-key 0
  - `BCM_FIELD_CMP_2` will compare the 32 msb of the TCAM result to comparison-key 2

11.19.1 SOC Properties

N/A

11.19.2 Configuration Flow

There are two modes for using the compare results:

Mode 1: Using the entire 6-bits result of a comparison.

In this mode the application can use one or more of the four compare results following qualifiers:

```
bcmFieldQualifyCompareKeysResult0
bcmFieldQualifyCompareKeysResult1
bcmFieldQualifyCompareKeysTcam0
bcmFieldQualifyCompareKeysTcam1
```

A helper function gives the offset of a specific compare operand inside the 6-bit chunk:

```
bcm_field_compare_operand_offset_get (init unit, int pair_id, bcm_field_comapre_operand_e
compare_operand , int *offset);
```

This utility function helps the user to set the entry value/mask according to the desired operand offset.

Example of flow:

1. Create a context on iPMF1 by calling:

   ```
   bcm_field_context_create(unit, stage=bcmFieldStageIPMF1,
                       context_info={context_compare_mode.compare_1_mode =
                       bcmFieldContextCompareTypeDouble}, &context_id);
   ```

   Note that this also creates the same context in iPMF2.

2. Create a compare Key by calling:

   ```
   bcm_field_context_compare_create(unit, stage=bcmFieldStageIPMF1, context_id, pair_id=1,
                                1st_key_info = {qual1, qual2,..}, 2nd_key_info  = {qual3, qual4})
   ```

3. Create a FG in iPMF2 stage by calling:

   ```
   bcm_field_group_add(unit, fg_info = { stage=bcmFieldStageIPMF2, qualfier_array = {
   bcmFieldQualifyCompareKeysResult0 ...} , action_array={TC,..}}
   ```

4. Attach the created field Group to context by calling:

   ```
   bcm_field_group_context_attach(unit, fg_id, context_id, attach_info = {qual_info={qual_type
   =bcmFieldQualifyCompareKeysResult0, input= metadata2 , offset = 0 , arg= 0} );
   ```

5. Call utility function to retrieve the desired result bit offset `bcm_field_compare_operand_offset_get(unit, pair_id=1,  bcmFieldCompareOperandNotEqual , &op_offset)`.

6. Add entry accordingly:

   ```
   bcm_field_entry_add (unit, fg_id, ent_id, key_info={qual_type =bcmFieldQualifyComapreResultKey0,
   val = 1<<&op_offset, mask=1<<&op_offset}, payload_info={type=bcmFieldActionTC, value= 3}).
   ```

Mode 2: A specific compare operand is used in the lookup, and the information is taken from specific pairs and specific operands per context.

This is done by using a 1-bit size user-defined qualifier.

The `bcm_field_group_qualifier_offset_get` is called to retrieve the qualifier offset in the key, as well as the new helper function `bcm_field_compare_operand_offset`. The sum of the qualifier offset to the operand offset, can be used in the field group context attach info, to indicate where to take the result from (which pair result, and which bit out of the result).

Example of flow:

1. Create a user-defined qualifier:
```
bcm_field_qualifier_create(unit, qual_info={stage=bcmFieldStageIPMF2, size=1, name =
                           "1bitCmpResult", &qual_id}
```
2. Create a context on iPMF1:
```
     bcm_field_context_create(unit, stage=bcmFieldStageIPMF1,
                        context_info={context_compare_mode.compare_1_mode =
                    bcmFieldContextCompareTypeDouble,context_compare_mode.compare_2_mode =
                        bcmFieldContextCompareTypeDouble}, &context_id);
```
3. Create two compare Keys:
```
bcm_field_context_compare_create(unit, stage=bcmFieldStageIPMF1, context_id, pair_id=1,
    1st_key_info = {qual1, qual2,..}, 2nd_key_info  = {qual3, qual4})
bcm_field_context_compare_create(unit, stage=bcmFieldStageIPMF1, context_id, pair_id=2,
    1st_key_info = {qual5, qual6,..}, 2nd_key_info  = {qual7, qual8})
```
4. Create two contexts in iPMF2 stage: ctx_1, ctx_2 by calling
```
bcm_field_context_create (unit, bcmFieldStageIPMF2, &ctx1/2);
```
5. Create a FG in iPMF2 stage:
```
bcm_field_group_add (unit, fg_info ={ stage=bcmFieldStageIPMF2, qualifier_array =
                     { qual_id, ...} , action_array={TC,...}}
```
6. Retrieve the desired result offset by calling:
```
bcm_field_group_qualifier_offset_get(unit, fg_id, qual_id, &qual_offset).
```
7. Call utility function to retrieve the desired result bit offset per pair:
```
bcm_field_compare_operand_offset_get(unit, 1,  bcmFieldCompareOperandNotEqual ,
&op1_offset).
bcm_field_compare_operand_offset_get(unit, 2, bcmFieldCompareOperandFirstKeyBigger , &op2_offset).
```
8. Attach FG to ctx1 the qualifier value will be taken from the bit not- equal from first pair result
```
bcm_field_group_context_attach(unit, fg_id, ctx_1, attach_info={qual_info={qual_type =qual_id,
input= metadata , offset = qual_offset+op1_offset , arg= 0} )
```
9. Attach FG to ctx2: the qualifier value will be taken from the bit first key bigger from second pair result:
```
bcm_field_group_context_attach(unit, fg_id, ctx_2, attach_info={qual_info={qual_type =qual_id,
input= metadata  , offset =  qual_offset+op2_offset, arg= 0} );
```
10. Add entry accordingly:
```
bcm_field_entry_add (unit, fg_id, ent_id, key_info={ qual_type="1bitCmpResult", qualifier_val = 1,
qualifier mask=1} , payload_info={ type=bcmFieldActionTC, value= }  ).
```

11.19.3 Application Reference

Using Compare Result

Example of using compare result in single/double mode

- **Type:** CINT example
- **Path:**

  `$SDK/src/examples/dnx/field/cint_field_compare_double.c`

  `$SDK/src/examples/dnx/field/cint_field_compare_single.c`

Using Compare Operand Offset Get

Example of using compare operand offset get helper function

- **Type**: CINT example
- **Path**:

  `$SDK/src/examples/dnx/field/cint_field_compare_full_offset_get.c`

  `$SDK/src/examples/dnx/field/cint_field_compare_specific_offset_get.c`

11.20 Exact Match Lookup

11.20.1 EXEM Field Group

- EXEM performs Exact Match (EM lookups).
  - It means an entry does not have a mask, unlike with TCAM, and the entire key must match (although the key size may vary) to have hit.
- EXEM is a part of the MDB. Not all the MDB profiles support EXEM.

Configuration is done in the same manner as in TCAM, with two following differences:

- When creating the FG, it is required to specify the FG type as Exact Match.
- When adding an entry, no entry handle is used and the mask field is not relevant.

For more details, see Section 11.20.1.2, Configuration Flow.

The EXEM interface is connected to the following stages:

- iPMF1 and ePMF can access large EXEM.
- Either iPMF2 OR iPMF3 can access small EXEM, as determined by the SOC property `pmf_sexem3_stage`.

Each EXEM entry consists of Key plus Payload. The maximum number of entries is defined by the MDB profile and the entry sizes (adjusted to the key and payload size provided). Therefore, the smaller the entries are, the more entries can be written.

The Large EXEM's key is limited to 160 bits, and its payload is limited to 55.

The small EXEM's key is limited to 80 bits, and its payload is limited to 56.

The hashing is performed only on a limited number of bits on the key: up to 80 LSBs of the key for large EXEM and 48 LSBs for small EXEM.

Also note that the exact matching uses hashing functions, so the number of entries that can be added is not necessarily the full space that was allocated to EXEM. The hashing is performed only on the 80 or 48 LSBs, so for larger entries, there is only a very limited number of entries with identical 80 or 48 LSB that can be added.

For example, in iPMF1, if qualifying upon the source and destination MAC addresses, plus the VLAN ID, the total bits are 108 bits, which is above 80. Because most of the changes are expected to be in the LSB half of the addresses and the VLAN ID, it might be best to break the MAC address qualifiers down to four user-defined qualifiers. Each qualifier takes either the LSB or MSB half of a MAC address. The qualifier taking the LSB half and the qualifier taking the VLAN ID are placed first in the field group and are located before the qualifiers taking the MSB half.

Also note that identical entries could collide even across different field groups if they are close in size.

11.20.1.1 SOC Properties

The SOC property `pmf_sexem3_stage` indicates whether exact match field groups are available on iPMF2 or iPMF3.

11.20.1.2 Configuration Flow

1. Add a new Exact Match FG: `bcm_field_group_add()`
   - Type = `bcmFieldGroupTypeExactMatch`
   - Total number of qualifiers must be according to the stage (as defined above).
2. Attach the Exact Match FG to the context using the API `bcm_field_group_context_attach()`
3. Add entries to the FG using `bcm_field_entry_add()`
   - Specify Exact Match FG ID
   - Specify the exact qualifiers value for the key to be matched. All qualifiers in the field group must be specified for each entry in the field group.
   - Specify the action values to be performed when a hit occurs.

11.20.1.3 Application Reference

Example of using EXEM Field group

- **Type**: CINT example
- **Path**: `$SDK/src/examples/dnx/elephant_flow/cint_field_exem.c`

11.20.2 Flush Machine

The flush machine allows EXEM to learn entries if a lookup miss occurs.

The main flush machine capabilities are as follows:

- Add an entry dynamically if the entry is not present in the EXEM database.
- Configure a flush profile, which holds lower and upper thresholds for aging.

NOTE: If entries are added statically by the user, they can also *age out* if aging was configured (that is, a flush profile was attached).

- Change the entry payload based on a set of rules, such as the following:
  - Key value
  - Payload value
  - Hit or miss
  - Threshold

  The payload for dynamic entry can be up to 32b length.

NOTE: Learned entries are not displayed on entry list diag.

11.20.2.1 Configuration Flow

1. The trace packet must be set. To set the trace packet, call `bcm_instru_gport_control_set()`.
   - `in_port`
   - `bcmInstruGportControlTraceProbability`
   - `probability`
2. To indicate where the flush profile is located in the learned payload (0:31 because the payload is up to 32b), call `bcm_switch_control_indexed_set()`.
   - Once per init of device. It cannot be changed per Field Group
   - `type`: `bcmSwitchExemFlushProfilePayloadOffset`
   - `index`: Should be 0
   - value: 0:31 offset of the flush profile in the FG payload
3. To enable flush learning per context, call `bcm_field_context_param_set()`.
   - `Field_stage`: Must be iPMF2 (only iPMF2 context can use flush learn)
   - `Context_id`: Context ID to enable flush learning for
   - `Input_type`: `bcmFieldContextParamTypeFlushLearnEnable`
   - `Param_arg`: None
   - `Param_value`: TRUE
4. Configure EXEM FG using `bcm_field_group_add()`.
5. Create a flush profile by using `bcm_field_flush_profile_create()`.
6. Configure the flush profile for the EXEM FG and attach it to the EXEMFG using `bcm_field_flush_profile_attach()`.

If dynamically changing the payload of learned entries is not required, the following step is not necessary.

7. Call `bcm_field_flush_entry_add()` to add a rule for the given FG that will dynamically change the payload of the entry when the rule is triggered.

11.20.2.2 Application Reference

Example of using EXEM learn capabilities.

- **Type:** CINT example
- **Path**: `$SDK/src/examples/dnx/elephant_flow/cint_elephant_flow.c`.
- **Path**: `$SDK/src/examples/dnx/elephant_flow/cint_elephant_flow_field.c`.

11.21 ACE – Egress PMF Extension

The ACE table is an exact match table that is part of MDB. It is located in the ETPP block and serves as an egress ACL extension.

The ACE table allows the egress-PMF to perform actions on ETPP (indirect access)

- The egress-PMF action is an ace pointer (key to access the ACE table)
- In ETPP, the ace pointer accesses the table.
- The result of the table can be managed dynamically.

The ACE key is 22-bits wide and can serve the following applications:

- MC-Replication
- Ace-Pointer

The table size depends on the MDB profile.

There are 60 result formats for ACE tables. The result is 110 bits wide, The ACE entry size is 110 bits:

- 6 bits for entry format
- 104 bits for data

There are eight dedicated EFES machines that parse the results data according to result format.

ACE configuration is done via dedicated APIs:

- To generate a new format of ACE data: `bcm_field_ace_format_add()`
- To add entries with a given format : `bcm_field_ace_entry_add()`

For an example of how to use those APIs for Egress-ACL, see Section 11.21.2, Configuration Flow.

For a list of actions that can be used for the ACE egress extension stage, see Section 11.31.3, Actions List.

Just like in Field Groups, the user indicates, per action, whether it has a valid bit (for the EFES). Unlike lookup-based field groups, an action without a valid bit is allowed, but all entries in the ACE format must have the same action.

NOTE: The use of ACE as an extension of the egress PMF is supported only for unicast packets.

11.21.1 SOC Properties

None

11.21.2 Configuration Flow

1. Call `bcm_field_ace_format_add()` to add create a new ACE format. Indicate:
   - unit – Device ID
   - Flags – To create ACE format with ID (`BCM_FIELD_WITH_ID`)
   - Action array – Array of BCM action types defining the actions to be performed by the ACE format.
   - Valid bit usage array – Array indicating for each action whether it used a valid bit or not.
   - input/output – `Ace_id` (input if the `BCM_FIELD_WITH_ID` flag is set).
2. Call `bcm_field_ace_entry_add()` to add entries with values to ACE format. Indicate:
   - unit – Device ID
   - Flags to create ACE format with ID/update existing (`BCM_FIELD_WITH_ID`/`UPDATE`)
   - `ace_id` – ACE format ID
   - Action array – Array of BCM action types (subset of the one of the ACE format). If an action does not have a valid bit, it must appear in all entries of the ACE format.
   - Value array per action type

   The function returns: `encap_id` – 22 bit (That can be used as action in ePMF)
   - Input – If the WITH_ID or UPDATE flags are set
   - Output – If none of the flags were set
3. Call `bcm_field_group_add()` where stage – `bcmFieldStageEgress`, include in the action array the following action: `bcmFieldActionAceEntryId`
4. Call `bcm_field_entry_add()` with action ACE pointer equals to the `encap_id` retrieved in Step 2.

11.21.3 Application Reference

Example of using egress PMF extension:

- **Type:** CINT example
- **Path:** `$SDK/src/examples/dnx/field/cint_field_ace.c`

11.22 Constant Field Group

In a constant Field Group (FG) there is no need to define a key because there are no classifiers. Certain actions with constant values are performed to all selected packets by the context.

11.22.1 SOC Properties

None

11.22.2 Configuration Flow

To create a constant FG:

1. If the user-defined actions needed for the field group were not already created, create them using `bcm_field_action_create()`.
   a. `size` must be 0.
   b. `prefix_size` is the size of the action.
   c. `prefix_value` is the value to be written to the action (raw value only).
2. Call `bcm_field_group_add` API where:
   a. `fg_type` = bcmFieldGroupTypeConst
   b. `nof_quals` must be equal to zero.
   c. No qualifiers are defined for the `qual_array`.
   d. All actions are with constant values (with size zero, and any value is provided as a prefix).
3. Call `bcm_field_group_context_attach`
4. No entries should be added (like in DE FG).

11.22.3 Application Reference

Example of using constant Field Group:

- **Type**: CINT example
- **Path**: `$SDK/src/examples/dnx/field/cint_field_group_const.c`

11.23 System Headers

System headers are headers used internally by the SDK. System headers are appended or removed from a packet according to the PMF. PMF decisions are executed by the LBP block.

There are two types of system headers:

- System headers built in the ingress
  - FTMH – Fabric TM Header. Transmitted from ingress to egress
  - TSH – Timestamp Header
  - Internal – Packet Processing Header. Propagated from IRPP to ETPP
  - UDH – User-defined Header. Built-in IRPP and propagated till ETPP
- System headers not built in the ingress
  - PTCH – Port Termination Control Header. Parametrize the in port of packet
  - ITMH – Ingress TM Header. Holds information for ITM, not parsed by PP in the ingress

Setting the System Header

PMF1,2 decides on which header to add based on the System Header Profile. The system header profile can be selected using two different methods:

- Per Context in iPMF1,2 by setting context parameter
- Per Field Group – Action in iPMF1,2 (this method overrides the former one)

For more information regarding setting the system header profile, refer to Section 11.23.2, Configuration Flow.

Setting Bytes to Remove

- Bytes to remove can be chosen by two different methods:
  - Per Context in iPMF2 by setting context parameter
  - Per Field Group – Action in iPMF3 (this method overrides the former one)
- Num Layers to remove (`bcm_field_packet_remove_layers_t`):
  - `bcmFieldPacketRemoveLayerOffset0` – Remove nothing
  - `bcmFieldPacketRemoveLayerOffset1`– Until first layer (layer 0).
  - `bcmFieldPacketRemoveLayerForwardingOffset0` – Until FWD layer
  - `bcmFieldPacketRemoveLayerForwardingOffset1` – Until FWD layer+1
- Bytes to remove – [0:127]
- Usually will not be used when packet is mirrored

Use Macro `BCM_FIELD_PACKET_STRIP` (`nof_layers_to_remove`, `bytes_to_remove`) for action encoding.

If iPMF3 does not remove the header, it is removed by ETPP termination Block.

Supported System Headers

Currently defined header profiles (`bcm_field_system_header_profile_t`):

- `bcmFieldSystemHeaderProfileFtmhTshPphUdh` – Ethernet port
- `bcmFieldSystemHeaderProfileFtmh`—FTMH only
- `bcmFieldSystemHeaderProfileFtmhhPph`
- `bcmFieldSystemHeaderProfileFtmhTsh`—ITMH port without PPH
- `bcmFieldSystemHeaderProfileFtmhTshPph`—ITMH port with PPH
- `bcmFieldSystemHeaderProfileNone`—Raw ports

Default Contexts and Preselectors

For correct Flow, default programs are defined in the last entries of context selection TCAM.

Feature	System Header Profile	Bytes to Remove (index)	Default Context ID	Presel ID
Ethernet Program	bcmFieldSystemHeaderProfileFtmhTshPphUdh	1	0	127

NOTE: ITMH context and `ITMH_PPH` contexts are part of the application example. More details can be found in Section 11.22.3, Application Reference.

11.23.1 SOC Properties

The `stacking_extension_enable` SOC property determines whether to add the 2-byte FTMH stacking extension. This is useful for adding more bytes for Ingress-Egress cascading

SOC Property Value	Meaning
0	FTMH stacking extension is disabled (default).
1	FTMH stacking extension is enabled.

NOTE: Stacking extension is not used for stacking applications. It is used only for field processor purposes.

11.23.2 Configuration Flow

1. To set System Header Profile per context:

   Call API: `bcm_field_context_param_set()`
   - `Field_stage`: Must be iPMF1 (only iPMF1 context can choose the profile)
   - `Context_id` : context ID to set its profile to
   - `Input_type` :bcmFieldContextParamTypeSystemHeaderProfile
   - `Param_arg`: none
   - `Param_value`: one of the system header profile (defined in `bcm_field_system_header_profile_t`)
2. To set System Header Profile per field group:

   Call API: `bcm_field_group_add()`
   - `Stage`: iPMF1/iPMF2
   - `FG_type`: Any
   - `FG_quals`: Any
   - `FG_action`: bcmFieldActionSystemHeaderProfile
     - Action wins over context decision
3. To remove headers:

   a. Per Context: Call API: `bcm_field_context_param_set()`
   - `Field_stage`: Must be iPMF2
   - `Context_id`: context to set its profile to
   - `Input_type`: `bcmFieldContextParamTypeSystemHeaderStrip`
   - `Param_arg`: none
   - `Param_value`: values encoded by `BCM_FIELD_PACKET_STRIP()`

b. Per Field Group Call API: `bcm_field_group_add()`
   - Stage: iPMF3
   - `FG_type`: Any
   - `FG_quals`: Any
   - `FG_action`: bcmFieldActionBytesToRemove
   - Value will be set by `bcm_field_entry_add()` using `BCM_FIELD_PACKET_STRIP()`
   - Action wins over context decision

11.23.3 Application Reference

Example of using system headers and using bytes to remove:

- **Type**: CINT example
- **Path**: `$SDK/src/examples/dnx/field/cint_field_sys_headers.c`

Example of an application that can be found in the ITMH+PPH-programmable reference application.

- **Type:** Default Application Reference
- **Path:** `$SDK/src/appl/reference/dnx/appl_ref_field_itmh_pph_init.c`

Example of using the FTMH stacking extension header to move more information from ingress to egress.

- **Type**: CINT example
- **Path**: `$SDK/src/examples/dnx/field/cint_field_udh_cascading.c`

11.24 LIF and Port Profiles

Some entities, such as LIF = logical interfaces and ports, can be combined in profiles. Those profiles can be used as qualifiers for context selection/entries, or actions.

11.24.1 LIF Profiles

Ingress and egress logical interfaces (InLIF and OutLIF) can be associated with profiles. For more information, see Section 7.5, LIF Profile Resources.

The number of bits used by PMF is set in the SOC property `pmf_in_lif_profile_nof_bits.`

LIF profiles can be set by calling the following APIs:

- `bcm_port_class_set()` using the port-class attribute `bcmPortClassFieldIngressVport` or `bcmPortClassFieldEgressVport` (for examples, see Section 11.23.3, Application Reference).
- `bcm_vlan_control_vlan_set()` using `control.if_class` for ETH_RIF

Then the `bcmFieldQualifyInVportClass` qualifier can be used for context selection, `bcmFieldQualifyInVportClass0/1` can be used as the entry qualifier, and `bcmFieldActionInterfaceClassVPort` can be used as for the action.

NOTE:

- PMF is not the only application using InLIF and OutLIF profiles.

  For more information, see Section 7.5, LIF Profile Resources.

11.24.2 Port Profiles

Ingress and Egress PP-ports or TM-ports can be associated with port profiles. Port profiles can be set by calling the `bcm_port_class_set` API using the port-class attributes shown in the following table.

Port Class Attribute	Profile
`bcmPortClassFieldIngressPMF1PacketProcessingPort`	PP-port class in Field stage iPMF1
`bcmPortClassFieldIngressPMF1PacketProcessingPortCs`	Context selection class for packet processing port in Field stage iPMF1
`bcmPortClassFieldIngressPMF3PacketProcessingPort`	PP-port in Field stage iPMF3
`bcmPortClassFieldIngressPMF3PacketProcessingPortCs`	Context selection class for packet processing port in Field stage iPMF3
`bcmPortClassFieldEgressPacketProcessingPort`	PP-port in Field stage ePMF
`bcmPortClassFieldEgressPacketProcessingPortCs`	Context selection class for packet processing port in Field stage ePMF
`bcmPortClassFieldIngressPMF1TrafficManagementPort`	TM-port class in Field stage iPMF1
`bcmPortClassFieldIngressPMF1TrafficManagementPortCS`	Context selection class for TM-port in Field stage iPMF1
`bcmPortClassFieldIngressPMF3TrafficManagementPort`	TM-port class in Field stage iPMF3
`bcmPortClassFieldIngressPMF3TrafficManagementPortCS`	Context selection class for TM-port in Field stage iPMF3
`bcmPortClassFieldEgressTrafficManagementPort`	TM-port class in Field stage ePMF
`bcmPortClassFieldIngressPMF1PacketProcessingPortGeneralData`	Extra constant to assign to a packet processing port, up to 32 LSB. Field stage iPMF1
`bcmPortClassFieldIngressPMF1PacketProcessingPortGeneralDataHigh`	Extra constant to assign to a PP-port, beyond the first 32 bits. Field stage iPMF1
`bcmPortClassFieldIngressPMF3PacketProcessingPortGeneralData`	Extra constant to assign to a PP-port, first 32 bits. Field stage iPMF3
`bcmPortClassFieldIngressPMF3PacketProcessingPortGeneralDataHigh`	Extra constant to assign to a PP-port, beyond the first 32 bits. Field stage iPMF3

Then `bcmFieldQualifyPortClassPacketProcessing` or `bcmFieldQualifyPortClassTrafficManagement` qualifiers can be used for context selection or entry qualifier, depending on whether the port profile is for PP or TM.

NOTE: When setting a port profile for trunk port, all its local ports will be set. For each port that is added to the trunk port afterwards, a port profile should be set.

11.24.3 Application Reference

Example of using LIF profile as a qualifier:

- **Type:** CINT example
- **Path:** `$SDK/src/examples/dnx/field/cint_field_inlif_profile.c`

Example of using port profile as a qualifier:

- **Type:** CINT example
- **Path:** `$SDK/src/examples/dnx/field/cint_field_port_profile.c`

11.25 Ingress Vlan Translate Stage – TCAM Field Group

In the ingress VLAN translate stage, it is possible to create a field group that replaces the VLAN translate TCAM interface configured on init (`IN_AC_TCAM_DB`).

The field group type `bcmFieldGroupTypeTcam` can be used for the `bcmFieldStageIngressVlanTranslation` stage with following restrictions:

- It replaces the interface of `bcmFieldAppDbVlanPort` in all L2 contexts
- It supports seven qualifiers in the following exact order:
  - `[0] = bcmFieldQualifyVpn`
  - `[1] = bcmFieldQualifyL4DstPort`
  - `[2] = bcmFieldQualifyL4SrcPort`
  - `[3] = bcmFieldQualifyIp4Protocol`
  - `[4] = bcmFieldQualifySrcIp`
  - `[5] = bcmFieldQualifyDstIp`
  - `[6] = bcmFieldQualifyVlanId`
- It supports only one action (IN_LIF) and should be one of the following, based on encoding:
  - `bcmFieldActionInVport0` – GPORT encoding
  - `bcmFieldActionInInterface0` – Interface encoding
  - `bcmFieldActionInVport0Raw` – No encoding; hardware raw value
- `bcm_field_context_attach()` should be invoked when context = `bcmFieldAppTypeL2`
  It is not necessary to call `bcm_field_context_create()` for the `bcmFieldStageIngressVlanTranslation` stage.

To add entries to the field group, use `bcm_field_entry_add()`, similar to how it is used with any other TCAM-type field groups.

NOTE:

- This configuration can be added only once per init because `bcm_field_group_delete()` and `bcm_field_group_context_detach()` do not work for this type of field group.
- The `field last info` command is not supported for this stage. Instead, use `pp vis ikleap`.

11.25.1 Application Reference

Example of using LIF profile as a qualifier:

- Type: cint example
- Path: `$SDK/src/examples/dnx/field/cint_field_vtt_stage.c`

11.26 Application Reference and Field Applications

To see full list of applications use the command `appl list`.

To run a certain application use the command `appl run id=x`.

11.26.1 Field Application Reference

11.26.1.1 ITMH

- File Name – `$SDK/src/appl/reference/dnx/appl_ref_itmh_pph_init.c`
- Context – ITMH Context
- Stage – iPMF2
- Resources Used – DE/FEM
- Short description – The ITMH application handles the TM header in Ingress stage.

This application is selected when PPH is not injected with ITMH. The selected context in iPMF1 will parse ITMH, and FTMH will be created instead. If ASE_OAM or TSH are injected, they are not parsed and keep the same values.

NOTE:

- If TSH is injected, Flow_ID extension is also triggered by this context.
- One exception is for packets injected with ASE_OAM and TSH. These packets are not supported by the application, and Flow_Id extension is not created.
- If ASE_1588 is injected (with or without TSH), it will be parsed. When the packet is injected with a skipped header, the FEM increment value for ASE_offset must be updated accordingly. (Relevant for the BCM88690, BCM88800, and BCM88830.)

11.26.1.2 ITMH PPH

- File Name – `$SDK/src/appl/reference/dnx/appl_ref_field_itmh_pph_init.c`
- Context – ITMH_PPH Context
- Stage – iPMF1/iPMF2
- Resources Used – Compare/DE/FEM
- Short description – The ITMH PPH application handles the TM header in the ingress along with the PPH extension

11.26.1.3 Field SRv6

File Name: $`SDK/src/appl/reference/dnx/appl_ref_srv6_field_init_deinit.c`

- Main function: `appl_dnx_field_srv6_init()`
- Context: Non-default contexts
- Stage: iPMF1/2/3(BCM88830 also ePMF)
- Resources used: TCAM and DE

This feature enables the following functionality for SRv6 flows:

- SRv6 Endpoint: (BCM88690, BCM88800, and BCM88480)
  - Copy the next SID to UDH
  - Set `parsing_Start_Type` to SRv6_Endpoint

- SRv6 Endpoint PSP
  - Same as Endpoint (BCM88690, BCM88800, and BCM88480)
  - Set TM header compensation
- SRv6 Endpoint micro-sid
  - Build next-hop IPv6 DIP in UDH
  - Set `parsing_Start_Type` to SRv6_Endpoint
  - Set UDH type to 0x55 (value assumed by the SDK)
- SRv6 USD (BCM88690, BCM88800, and BCM88480)

  Calculates the number of bytes to cut, per number of SIDs on SRV6 extension. Set it on `parsing_start_offset`.
- SRv6 extended termination first pass (BCM88830)
  - Estimate bytes to remove in egress pipeline, using ePMF.
  - Set UDH data (using the iPMF) to indicate the size of bytes to remove.

11.26.1.4 Field RCH

File Name: `$SDK/src/appl/reference/dnx/appl_ref_field_rch_handle_init.c`

- Main function: `appl_dnx_field_rch_handle_init()`
- Context: Non-default contexts
- Stage: iPMF1/2/3
- Resources used: DE

This feature enables the following functionality RCY ports, RCH-based:

- All ports but extended encapsulation

  Configure IRPP to crop RCH.
- Extended encapsulation port
  - Keep RCH to be cropped at Egress
  - Set `parsing_Start_type` to RCY
  - Set `parsing_start_offset` to 0
  - Set `forward_layer_index` to 1

11.26.1.5 Field FTMH MC

This application is selected when a packet is injected through a recycle port (a port profile must be set with port class `bcmPortClassFieldIngressPMF1PacketProcessingPortCs` and class ID 0x7). The selected context in iPMF1 parses the FTMH and LB-key extension.

For more information, refer to the Multicast chapter in the *Traffic-Management* programming guide (88690-PG2xx).

- File Name: `$SDK/src/appl/reference/dnx/appl_ref_field_ftmh_mc_init.c`
- Context: FTMH MC Context
- Stage: iPMF2
- Resources Used: DE
- Short description: The FTMH MC application handles the Fabric TM header in Ingress stage for two-pass MC traffic.

11.27 Diagnostics

To view the most recent diagnostics for the field module, type `field` in the BCM shell.

11.27.1 Field Group Diagnostics

Two types of diagnostics are available.

1. **List:** Give basic information about all Field Groups configured in the device, such as:
   - FG ID
   - Name
   - Type
   - Stage
   - Key size
   - Payload size

   Shell command: `field group list`
2. **Detailed info**: Gives detailed information about a specific field group or range of field groups like:
   - FG ID
   - Name
   - Type
   - Stage
   - Key size
   - Payload size
   - Access profile – Only TCAM FG
   - Bank IDs – Only TCAM
   - Contexts for FG – Shows all contexts to current FG
     - Context ID
     - Name
   - Key and Payload Info
     - Qualifier
     - Qualifier size
     - Qualifier LSB
     - Action
     - Action size
     - Action LSB
   - FG Entry Info – Only TCAM
     - Banks
     - Number of entries – Total number of entries and per bank

   Shell Command: `field group info group=X`

11.27.2 Context Diagnostics

11.27.2.1 List

Gives basic information about a specific field context or range of field contexts like:

- CTX ID
- Stage
- Name
- Presel (qualifier type and Value)
- Mode – Relevant only for IPMF1 and IPMF2

11.27.2.2 Detailed Info

Gives detailed information about a specific field context on a specific field stage like:

- CTX ID
- Stage
- Name
- Presel (Value)
- Stage
- Mode – Relevant only for IPMF1 and IPMF2
- Attached FGs and keys, which are being allocated for every FG.
- Cascading From – Relevant only for IPMF2
- If in CMP mode: CMP Key info – For both pairs and keys
  - Qual
  - Size
  - Lsb
  - Type
  - Arg
  - Offset

11.27.3 Attach Diagnostics

Detailed: Gives Field group attach information for a specific FG and context (given by the user):

- FG ID
- FG Name
- CTX ID
- CTX Name
- Key info:
  - Qual type
  - Qual size
  - Qual LSB
  - Input type
  - Input arg
  - Offset

- Action info:
  - Type
  - Size
  - LSB
  - Priority
- FES info
  - ID
  - Action type

Input parameters:

- Group (mandatory) – Should be a specific FG ID (0-127). If the FG is not being specified an error will be printed.
- Context-Id (mandatory) – Should be a specific Context ID (0-63). If the context is not being specified an error will be printed.

11.27.4 TCAM Bank Diagnostics

TCAM bank info: Show information per TVAM bank:

- Bank-ID
- Bank Owner (stage)
- Number of free 80-bit entries
- Number of free 160-bit entries
- Number of free 320-bit entries
- Field groups which have entries in this bank. per Field group:
  - Field group name
  - Field group ID
  - TCAM handler ID
  - Key size.
  - Number of entries for this field group installed in the bank

11.27.5 TCAM Management

TCAM management diagnostics shows information about TCAM bank management such as:

- Handler ID
- Key Size
- Action Size
- Stage
- Prefix Size
- Prefix Value
- Access Profile Number
- Bank Allocation Mode
- Bank IDs
- Number of entries per Bank

List: Shows a list of TCAM bank handlers configured in the system.

Info: Show details about specific TCAM handler. Input: TCAM handler-Id.

11.27.6 Entry Diagnostics

11.27.6.1 List

Shows per Field Group:

- Fg-ID
- Fg-name
- TCAM bank Id
- Number of free entries
- Used entry IDs (for TCAM FG)

Entry list diagnostics are not supported for direct table TCAM.

11.27.6.2 Info

Shows detailed information for specific entry

Entry Info:

- Fg-ID
- Entry-ID
- TCAM bank-ID
- Bank offset (location in the bank)
- key and payload Info:
  - key info: qualifier type, size, LSB, qualifier value, and mask.
  - payload Info: action type, size, LSB, action value.

11.27.6.3 FEM Diagnostic

Displays SW state and HW information.

- Field FEM display `by_fg`
  - FEM display for a range of FEMs. Range is specified using `group`, `sec_group` and `fem id`
  - If no `group` is specified then `0-nof_fgs-1` is assumed (0-127).
  - If no `sec_group` is specified then `0-nof_fgs-1` is assumed (0-127).
  - If no `active_actions` is specified then it is ignored.
  - If no `fem` is specified then `0-nof_fem_id-1` is assumed (0-15).
  - If only one value is specified for `group` or `sec_group` then this single value range is assumed.
  - If `valid` is not specified or is set to `yes` then only `group` or `sec_group` combinations, with at least one allocated FEM, are displayed.
- Field FEM display `by_context`
  - FEM display for a range of FEM. Range is specified using `fem id` and `context`
  - If no context is specified then minimal val-maximal val is assumed (0-63).
  - If no `fem` is specified then `0-nof_fem_id-1` is assumed (0-15).
  - If only one value is specified for context or FEM then this single value range is assumed.
  - If `valid` is not specified or is set to `yes` then only contexts, with at least one allocated fem, are displayed.

11.27.7 Qualifiers

List: Shows information about predefined and user-defined qualifiers configured input parameters.

11.27.8 Actions

List: Shows information about predefined and user-defined actions configured in the system.

11.27.9 ACE Diagnostics

11.27.9.1 ACE Format Diagnostics

List: Shows information about all ACE formats: name, payload size, efes mask usage.

Information: Shows the actions that comprise specific ACE format.

- Input: ACE-format-ID
- Output: list of actions, per action: size, LSB, valid bit

11.27.9.2 ACE Entries Diagnostics

Information: Shows the action values per specific ACE-format.

- Input: ACE-format-ID
- Output: list of actions, per action: size, LSB, value, valid bit

11.27.10 System Header Profiles

Shell command: Field `sys_headers info`:

Shows list of system header profiles configured in the system, with following attributes:

- profile-id
- build-pph
- build-ftmh
- build-tsh
- build-udh

11.27.11 EFES Diagnostics

- List – Input: EFES program
  Lists all contexts and field groups that use a specific EFES ID, and their EFES program ID (hardware)
- List – Input: EFES masks
  Lists all contexts and field groups that use a specific EFES mask ID, and their EFES program ID (hardware)
  Lists the values of the EFES masks.
- List – Usage
  Lists the number of EFES programs and masks used by each EFES ID (hardware)
- Information: Detailed information per EFES ID and EFES program ID

11.27.12 Field System View

This diagnostic view shows a general system view per stage:

Per valid (configured Presel-ID):
- Presel qualifiers set for that preselector (type, value, argument, and mask).
- Context which this presel is pointing (context-ID and name).
- Field Group which attached to this context (FG ID and name).

11.27.13 Field Last

Shell command: `field last info`.

This diagnostic command shows information about field configurations and performed lookups per stage or range of stages.

This command is functional only if one packet was injected; otherwise, it might return incorrect information.

The Entry ID column of the Field Group Info table displays N/A in following cases:
- If the option `one_pkt` is not specified.
- If it is not relevant to the presented field group type.

TCAM DT field groups are not supported in this diagnostic command.

11.28 API Descriptions

11.28.1 bcm_field_group_*

11.28.1.1 bcm_field_group_add()

```
bcm_field_group_add(
   int unit,
   uint32 flags,
   bcm_field_group_info_t * fg_info,
   bcm_field_group_t * fg_id)
```

Parameters:
- unit: [IN] device ID
- flags: [IN]
  - `BCM_FIELD_FLAG_WITH_ID` – To add a field group with ID. If this flag is set `fg_id` considered as input otherwise as output
- fg_info: [IN]
  - `field_stage` – For which stage the field group is created, which is one of the following.
  - `fg_type` – Defines the type of lookup and hardware type that will hold the add entries. (TCAM/Direct Extraction...), which is one of the following.
  - `nof_quals` – Number of valid qualifiers
  - `qual_types` – Array of `bcm_field_qualify_t` to define how the lookup key is built. The order of array determine the order of qualifiers in the key.
  - `qual_is_ranged` – For external TCAM field groups, indicates that the corresponding qualifier accepts a range of value for a single entry instead of a value and a mask.

- nof_actions – Number of valid actions
- action_types – Array of bcm_field_action_t to define how the lookup result is built. The order of each array will determine the order of actions in the lookup result.
- action_with_valid_bit – Indicated that the corresponding action added a valid bit as its LSB.
- name – String for application use.
- tcam_info – Used for TCAM field groups, to indicate how many banks to reserve.
  - nof_tcam_banks – Numbers of bank to pre-allocate
  - tcam_bank_ids – Array of the bank IDs to pre-allocate (used only when bank_allocation_mode is not AUTO)
  - core_ids – Array of the core IDs for each bank ID (initialized to BCM_CORE_ALL)
  - bank_allocation_mode – There are three available modes (bcm_field_tcam_bank_allocation_mode_t):
    - Auto mode: The TCAM module manages the banks allocated by the field group automatically without user interference. When an additional bank is required while adding a new entry, the TCAM module selects the appropriate available bank and allocates it.
    - Select mode: The user is responsible for allocating the banks for the field group by either supplying a list of banks to pre-allocate when adding the field group (see the previous parameters) or by using the function bcm_field_tcam_bank_add(). Adding a new entry while the field group does not have the enough resources results in an error.
    - Select with Location (Entry Add Location [EAL]) mode: As in the previous mode, the user is responsible for allocating the banks for the field group. Additionally, the user mut specify the exact absolute location for each added entry by providing the exact location with the priority parameter in the bcm_field_entry_add() API.

- fg_id – Handler-ID for the created field group.

11.28.1.2 bcm_field_group_info_get()

```
bcm_field_group_info_get(
   int unit,
   bcm_field_group_t fg_id,
   bcm_field_group_info_t * fg_info)
```

Parameters:

- unit: [IN] device ID
- fg_id: [IN] handler-Id for the created field group.
- fg_info: [OUT]. See details of the struct in Section 11.28.1.1, bcm_field_group_add().

11.28.1.3 bcm_field_group_delete()

```
bcm_field_group_delete(
   int unit,
   bcm_field_group_t fg_id)
```

Parameters:

- unit: [IN] device ID
- fg_id: [IN] handler-Id for the deleted field group.

NOTE: The field group is not deleted and an error is returned if the field group is attached to a context or has installed entries (except for a TCAM field group, where all entries are deleted when deleting the field group).

11.28.1.4 bcm_field_group_qualifier_offset_get()

```
bcm_field_group_qualifier_offset_get(
   int unit,
   uint32 flags,
   bcm_field_group_t fg_id,
   bcm_field_qualify_t bcm_qual,
   int *offset)
```

Parameters:

- `unit`: [IN] device ID
- `flags`: [IN] future use
- `fg_id`: [IN] handler-Id for the field group.
- `bcm_qual`: [IN] The BCM qualifier to return the offset
- `offset`: [OUT] The offset for the given qualifier for the given `fg_id`

NOTE: This API is usually used for FEM and `FES_action_add` APIs.

11.28.1.5 bcm_field_group_action_offset_get()

```
bcm_field_group_action_offset_get(
   uint32 flags,
   bcm_field_group_t fg_id,
   bcm_field_action_t bcm_action,
   int *offset)
```

Parameters:

- `unit`: [IN] device ID
- `flags`: [IN] future use
- `fg_id`: [IN] handler-Id for the field group
- `bcm_action`: [IN] The BCM action to return the offset
- `offset` [OUT]: The offset for the given action for the given `fg_id`

NOTE: This API is usually used for cascading from iPMF1 to iPMF2.

11.28.1.6 bcm_field_group_context_attach()

```
bcm_field_group_context_attach(
   int unit,
   uint32 flags,
   bcm_field_group_t fg_id,
   bcm_field_context_t context_id,
   bcm_field_group_attach_info_t * attach_info_p)
```

Parameters:

- `fg_id` [IN]: FG ID to attach context to, after attach API context will also perform the lookup configured in the FG
- `flags` [IN]: Future use
- `context_id`: Context ID to attach to FG.

- attach_info [IN]: holds detailed info how context defines each qualifier/action of the FG
  - key_info
    - nof_quals – Number of elements in the arrays qual_types and qual_info
    - qual_types – Qualifier array should be in the same order as configured in bcm_field_group_add()
    - qual_info:

      Input type – See the description in Input Types

      Input arg – See the description in Input Argument

      Offset – The offset of the qualifier inside the input_type buffer. See Offset.
  - payload_info
    - nof_actions – Number of elements in the arrays action_types and action_info.
    - action_types – Action array should be in same order as configured in bcm_field_group_add.
    - action_info:

      Priority – The same action can be repeated for specific context (context was attached to two FGs that share the same action). Priority is set to determine which action will *win*, where the higher the number the higher the priority. BCM_FIELD_ACTION_INVALIDATE can also indicate that the action should not be performed (effectively turning it into a void action for a specific context).

      Polarity – Relevant only for Direct Extraction FGs to determine if the condition bit equals 0 or 1 (that is, the action is valid when the value is 0/1)
    - nof_invalidate_fems – Number of elements in the array invalidate_fems
    - invalidate_fems – A pointer initialized to NULL. If nof_invalidate_fems is larger than zero, it is treated as an array of FEM positions. Each FEM used by the field group listed in this array is not configured for this context. The caller is expected to provide the memory where this array resides.
  - compare_id – Indicates which compare pair to use during comparison.

Input Types

The base of qualifier indicate from which part of the PBUS the info is taken, relevant for all type of qualifiers.

- bcmFieldInputTypeLayerFwd – Takes data from a layer in the header, using the forwarding layer information. Not supported on the iPMF3 stage.
- bcmFieldInputTypeLayerAbsolute – Takes data from a layer in the header, ignoring forwarding layer information. Not supported on the iPMF3 stage.
- bcmFieldInputTypeMetaData – Takes data from the Metadata.
- bcmFieldInputTypeLayerRecordsFwd – Takes data about a layer in the header (layer record), using the forwarding layer information.
- bcmFieldInputTypeLayerRecordsAbsolute – Takes data about a layer in the header (layer record), ignoring the forwarding layer information.
- bcmFieldInputTypeCascaded – Takes data from the result of a field group from an earlier stage, using cascading, such as the result of iPMF1 at iPMF2 or the result at Ingress PMF stages. Specifically, iPMF2 can qualify on the result of an iPMF1 lookup.
- bcmFieldInputTypeConst – Writes a constant value. Can only write a zero value in ePMF. In ePMF the entire lookup key cannot be composed solely of qualifiers with input type const.

Input Argument

Input arg has different meaning per input type as follows:

- bcmFieldInputTypeLayerFwd – How many layer to increment from the forwarding layer (i.e. FWD+input_arg)
- bcmFieldInputTypeLayerAbsolute – How many layer to increment from the absolute layer (i.e. 0+input_arg)
- bcmFieldInputTypeMetaData – Has no meaning

- `bcmFieldInputTypeLayerRecordsFwd` – How many layer to increment from the forwarding layer (i.e. FWD+input_arg)
- `bcmFieldInputTypeLayerRecordsAbsolute` – How many layer to increment from the absolute layer (i.e. 0+input_arg)
- `bcmFieldInputTypeCascaded` – Field group Id of the 'Cascaded from' result
- `bcmFieldInputTypeConst` – A 32 bit constant.

To illustrate the difference between FWD and absolute input types, if the forwarding layer is 3 and the input argument is 1, then an absolute input type refers to layer index 1, while a relative input type refers to layer index 4. Be aware that the input argument for layer and layer record input types is cyclical based on the number of layer indexes in the stage. If there is a 5-layer index in the stage, a FWD input type with input argument 2 and forwarding layer 3 will give the layer index of 0.

Note that the default value for the input argument is 0.

Offset

The offset depends on the input type:

- `bcmFieldInputTypeLayerFwd` and `bcmFieldInputTypeLayerAbsolute` – The offset of the header inside the layer.
- `bcmFieldInputTypeMetaData` – Has meaning only in case of User Defined qualifier (good to use with `bcm_field_qualifier_info_get()` – The offset returned in the API is the offset of the qualifier in metadata).
- `bcmFieldInputTypeLayerRecordsFwd` and `bcmFieldInputTypeLayerRecordsAbsolute` – Offset in the LR in bits.
- `bcmFieldInputTypeCascaded` – Offset in the payload of FGs, this qualifier is cascaded from. (It is good to use this with `bcm_field_group_action_offset_get()`).
- `bcmFieldInputTypeConst` – No meaning.

So if using a metadata or layer record predefined qualifier, the offset will be completely ignored. If using a header input type, it is expected to indicate the offset of the header within the layer (because of this header input type can be shifted to read not from the beginning of the field that they are supposed to read, but note that the number of bits read will remain unchanged).

If using a user-defined qualifier, it is expected to provide the offset from where the input type begins (PBUS for metadata, layer for header, layer record for layer record, payload for cascaded, and so on). For example, to read starting from the second bit of a specific field, we will need to obtain the offset using the relevant API (`bcm_field_qualifier_info_get()`,`bcm_field_group_action_info_get()`, and so on) and then add one to it.

NOTE: The default value for the offset 0. For cases such as the VLAN header (802.1Q header) that default offset is almost certainly false.

11.28.1.7 bcm_field_group_context_info_get()

```
bcm_field_group_context_info_get(
   bcm_field_group_t fg_id,
   bcm_field_context_t context_id,
   bcm_field_group_attach_info_t * attach_info_p)
```

Parameters:

- `unit` [IN]: Device ID
- `fg_id` [IN]: FG ID
- `context_id` [IN]: Context ID.
- `attach_info` [OUT]: holds detailed information. See Section 11.28.1.6, bcm_field_group_context_attach().

NOTE: If invalidated FEMs are on the context, they are written to the memory pointed to by `attach_info.payload_info.invalidate_fems`. The caller is expected to provide this memory. Initialize the attach information before calling `bcm_field_group_context_info_get()` (by calling `bcm_field_group_attach_info_t_init()`), or to set `attach_info.payload_info.invalidate_fems` to NULL.

11.28.1.8 bcm_field_group_context_detach()

```
bcm_field_group_context_detach(
   int unit,
   bcm_field_group_t fg_id,
   bcm_field_context_t context_id)
```

Parameters:

- `unit` [IN]: Device ID
- `fg_id` [IN]: FG ID.
- `context_id` [IN]: Context ID.

11.28.1.9 bcm_field_group_cache()

```
bcm_field_group_cache(
   int unit,
   uint32 flags,
   bcm_field_group_t  fg_id,
   bcm_field_group_cache_mode_t mode)
```

Parameters:

- `unit` [IN]: Device ID
- `flags` [IN]: TBD
- `fg_id` [IN]: FG ID
- `mode` [IN]: The mode to which mode the field group is set.
  - `bcmFieldGroupCacheModeStart` – Start cache mode.
  - `bcmFieldGroupCacheModeInstall` – Install all pending entries to HW.
  - `bcmFieldGroupCacheModeClear` – Delete all pending entries from cache.

NOTE: Clear and install actions disable the cache mode. To enable cache mode after performing a clear or install, start must be invoked again.

11.28.2 bcm_field_context_*

11.28.2.1 bcm_field_context_create()

```
bcm_field_context_create(
    int unit,
    uint32 flags,
    bcm_field_stage_t stage,
    bcm_field_context_info_t * context_info,
    bcm_field_context_t * context_id)
```

Parameters:

- `unit`: [IN] device Id
- `flags`: [IN] BCM_FIELD_WITH_ID – Add a field context with ID. If this flag is set, `context_id` is considered as input; otherwise, it is considered as output
- `stage`: [IN] For which stage the field context is created (iPMF1/2/3, ePMF).
- `context_info` [IN]:
  - `context_compare_modes` – Defines which compare types are set for this context, for each compare operation (compare1, compare 2): none/single/double.
  - `cascaded_from` – The context-Id which the current context is cascaded from (relevant to iPMF2 stage context).
  - `hashing_enable` – Enable if a hash key is set on this context.
  - `Name` – String for application use.
- `context_id`: [INOUT] ID handler for the created context.

11.28.2.2 bcm_field_context_info_get()

```
bcm_field_context_info_get(
    int unit,
    bcm_field_stage_t stage,
    bcm_field_context_t context_id,
    bcm_field_context_info_t * context_info)
```

Parameters:

- `unit`: [IN] device Id
- `stage`: IN] For which stage the field context is created (iPMF1/2/3, ePMF.)
- `context_id`: [IN] Id handler for the context
- `context_info` [OUT]: For details of the struct, see Section 11.28.2.1, bcm_field_context_create().

11.28.2.3 bcm_field_context_destroy()

```
bcm_field_context_destroy(
    int unit,
    bcm_field_stage_t stage,
    bcm_field_context_t context_id)
```

Parameters:

- `unit`: [IN] device Id
- `stage`: [IN] For which stage the field context is created (iPMF1/2/3, ePMF.)
- `context_id`: [IN] ID handler for the destroyed context

NOTE: The context will not be deleted if has FGs attached to it.

11.28.2.4 bcm_field_context_param_set(

```
bcm_field_context_param_set(
    int unit,
    uint32 flags,
    bcm_field_stage_t bcm_stage,
    bcm_field_context_t context_id,
    bcm_field_context_param_info_t * context_params)
```

Parameters:

- `unit` [IN] Unit ID
- `flags` [IN] Future use (reserved)
- `stage` [IN] The stage of the context which the attribute is set.
- `context_id` [IN] Context ID to set the parameters to.
- `context_params` [IN]:
  - `param_type` – The parameter type
  - `param_arg` – Parameter argument
  - `param_val` – Parameter value

11.28.2.5 bcm_field_context_hash_create()

```
bcm_field_context_hash_create(
   int unit,
    uint32 flags,
    bcm_field_stage_t stage,
    bcm_field_context_t context_id,
    bcm_field_context_hash_info_t * hash_info)
```

Parameters:

- `unit` [IN]: Unit ID
- `flags` [IN]: Can be `BCM_FIELD_FLAG_UPDATE`, only `hash_function` and `hash_config` (including `additional_hash_config`) can be updated. `key_info`, `compression`, and `order` must keep the same values.
- `stage` [IN]: The stage of the context which the attribute is set.
- `context_id` [IN]: Context ID on which the HASH should be created.
- `hash_info` [IN]: struct
  - `key_info` – The information for the key
    - `qual_types` – Indicates which qualifier types construct the hash key
    - `qual_info`: (the same as in bcm_field_group_context_attach())
      - `input type` – As description in Input Types
      - `input arg` – As description in Input Argument
      - `offset` – The offset of the qualifier inside the input type buffer (see Offset).
  - `compression` – Enable or disable compress data
  - `order` – Enable or disable order data
  - Hash function – Hashing polynomial for the hashing mechanism
  - `hash_config` – Hashing configuration parameters describing which hash action is performed and on which signals
    - `action_key` – Key to config
    - `function_select` – Hashing function to perform on key. For supported values, see Table 6, Hash Function on Key
    - `crc_select` – If the hashing function selected is one of the augmentation functions, `crc_select` selects the CRC functions to be performed on the augmentation output. Each hash config must have a different `crc_select`.
  - `additional_hash_config`: Array of additional `hash_config` struct. Can be filled as a continuation of `hash_config`.
  - `nof_additional_hash_config` – Number of additional hash config data.

11.28.2.6 bcm_field_context_hash_info_get()

```
bcm_field_context_hash_info_get(
   int unit,
   uint32 flags,
   bcm_field_stage_t stage,
   bcm_field_context_t context_id,
   bcm_field_context_hash_info_t *hash_info)
```

Parameters:

- `unit` [IN]: Unit ID
- `flags` [IN]: Future use (reserved).
- `stage` [IN]: The stage of the context which the attribute is set.
- `context_id` [IN]: Context ID for which to get the hash key configuration.
- `hash_info` [OUT]: The same struct as above see Section 11.28.2.5, bcm_field_context_hash_create( ).

11.28.2.7 bcm_field_context_hash_destroy()

```
bcm_field_context_hash_destroy(
   int unit,
   bcm_field_stage_t stage,
   bcm_field_context_t context_id)
```

Parameters:

- `unit` [IN]: Unit ID
- `stage` [IN]: The stage of the context which the attribute is set.
- `context_id` [IN]: Context ID on which the HASH should be destroyed.

11.28.2.8 bcm_field_context_compare_create()

```
bcm_field_context_compare_create(
   int unit,
   uint32 flags,
   bcm_field_stage_t stage,
   bcm_field_context_t context_id,
   uint32 pair_id,
   bcm_field_context_compare_info_t *cmp_info)
```

Parameters:

- `unit` [IN]: Device ID
- `flags` [IN]: Future use (reserved)
- `stage` [IN]: The Field stage for which the compare should be done (iPMF1, 2, 3 and EPMF)
- `context_id` [IN]: Context ID to get context key and configuration mode
- `pair_id` [IN]: The pair ID to set for comparison
- `cmp_info` [IN]: Structure holds detailed info how the pair compare key is constructed
  - `first_key_info` – The information for the first key
    - `qual_types` – The qualifier types that construct the compare key
    - `qual_info`: The same as in bcm_field_group_context_attach().
      – input type – See Input Types

  - input arg – See Input Argument
  - offset – The offset of the qualifier inside the `input_type` buffer. See Offset.
- second_key_info – The information for the second key
  - `qual_types` – The qualifier types that construct the compare key
  - `qual_info`: The same as in bcm_field_group_context_attach()
    - input type – See Input Types
    - input arg – See Input Argument
    - offset – The offset of the qualifier inside the `input_type` buffer. See Offset.

11.28.2.9 bcm_field_context_compare_info_get()

```
bcm_field_context_compare_info_get(
   int unit,
   uint32 flags,
   bcm_field_stage_t stage,
   bcm_field_context_t context_id,
   uint32 pair_id,
   bcm_field_context_compare_info_t *cmp_info)
```

Parameters:

- `unit` [IN] – Device ID
- `flags` [IN] – Future use (reserved)
- `stage` [IN] – For which Field stage compare should be done (iPMF1, 2, 3 and EPMF)
- `context_id` [IN] – Context ID to get context key and configuration mode
- `pair_id` [IN] – The pair ID to set for comparison
- `cmp_info` [OUT] – The same struct as described in bcm_field_context_compare_create()

11.28.2.10 bcm_field_context_compare_destroy()

```
bcm_field_context_compare_destroy(
   int unit,
   bcm_field_stage_t stage,
   bcm_field_context_t context_id,
   uint32 pair_id)
```

Parameters:

- `unit` [IN]: device ID
- `flags` [IN]: future use (reserved)
- `stage` [IN]: For which Field stage compare is done (iPMF1, 2, 3 and EPMF)
- `context_id` [IN]: Handler of Context ID
- `pair_id` [IN]: which pair ID to destroy

11.28.3 bcm_field_presel_*

11.28.3.1 bcm_field_presel_set()

```
bcm_field_presel_set(
   int unit,
   uint32 flags,
   bcm_field_presel_entry_id_t * entry_id,
   bcm_field_presel_entry_data_t * entry_data);
```

Parameters:

- `unit` [IN]: Device ID
- `flags` [IN]: Currently not in use
- `entry-id` [IN]: Preselection ID structure:
  - stage: Context selection stage
  - presel-id: Preselector ID (range: 0 to 127)
- `entry-data`[IN]: Preselection data structure:
  - `valid` : Boolean indicates if this entry is valid
  - `context_id`: The context this presel ID will be pointing.
  - `nof_qualifiers`: Number of valid qualifiers the preselector entry
  - `qualfiers_data`: Array of qualifiers data structure
    - `qualfier_types`:

      `qual_argument` (relevant only for some of the qualifiers, such as the following two examples):

      For `ForwardingType` qualifier, `qual_argument` specifies which layer relative to the FWD layer to retrieve the type from. For example, `qual_argument=-1` means that the type is retrieved from layer FWD-1

      For `VPortRangeCheck`, `qual_argument` specifies which `OUT_LIF_RANGE` to consider (either 0 or 1)

      For a list of all relevant qualifiers, see Section 11.31.1, CS Qualifiers.

      `qual_value`

      `qual_mask`

11.28.3.2 bcm_field_presel_get()

```
bcm_field_presel_get(
   int unit,
   uint32 flags,
   bcm_field_presel_entry_id_t * entry_id,
   bcm_field_presel_entry_data_t * entry_data);
```

Parameters:

- `unit` [IN]: Device ID
- `flags` [IN]: Currently not in use
- `entry-id` [IN]: Preselection ID structure:
  - `stage`: Context selection stage
  - `presel-id`: preselector ID (range: 0 to 127)
- `entry-data`[OUT]: Preselection data structure as described in bcm_field_presel_set().

11.28.4 bcm_field_qualifier_*

11.28.4.1 bcm_field_qualifier_create()

```
bcm_field_qualifier_create(
   int unit,
   uint32 flags,
   bcm_field_qualifier_info_create_t * qual_info,
   bcm_field_qual_t * qualifier_id);
```

Parameters:

- `unit` [IN]: device ID
- `flags` [IN]: `BCM_FIELD_WITH_ID` the qualifier-ID should be explicitly indicated
- `qual_info` [IN]: structure holds detailed info of the qualifier to be created
  - `size`: Size of the qualifier to set in the key in bits
  - `name`: String for application use.
- `qualifier_id` [IN/OUT]: Handler-ID for the created qualifier. If the `BCM_FIELD_WITH_ID` flag is set is considered as input otherwise as output.

11.28.4.2 bcm_field_qualifier_info_get()

```
bcm_field_qualifier_info_get(
   int unit,
   bcm_field_qual_t  qual_id,
   bcm_field_stage_t stage,
   bcm_field_qualifier_info_get_t * qual_info);
```

Parameters:

- `unit` [IN] – Device ID
- `qualifier_id` [IN] – Handler-ID for the qualifier
- `stage [IN]` – The stage of the qualifier. Only relevant for predefined qualifiers.
- `qual_info [OUT]` –A structure that holds detailed information about the qualifier.
  - `qual_class` – The type of the qualifier. Indicates the possible input types it can receive (for example, metadata, header, layer records, or user defined).
  - `size` – Size of the qualifier in the key in bits.
  - `offset` – The offset associated with the qualifier. For example for a predefined metadata qualifier, this is the offset for the metadata field. For a predefined header qualifier, this is the offset inside the header.
  - `name` – The name of a predefined qualifier or the name provided for a user-defined qualifier.

11.28.4.3 bcm_field_qualifier_destroy()

```
bcm_field_qualifier_destroy(
   int unit,
   bcm_field_qual_t  qual_id);
```

Parameters:

- `unit` [IN]: device ID
- `qualifier_id` [IN]: handler-ID for the qualifier

11.28.4.4 bcm_field_qualifier_value_map()

```
bcm_field_qualifier_value_map(
   int unit, bcm_field_stage_t stage,
   bcm_field_qual_t  qual_id,
   uint32 bcm_value[],
   uint32 hw_value[]);
```

Parameters

- `unit` [IN]: Device ID
- `stage` [IN]: The stage the qualifier is defined for
- `qualifier_id` [IN]: The handler-ID for the qualifier
- `bcm_value` [IN]: The BCM value to map
- `hw_value` [OUT]: The mapped HW value of the given bcm value for the given qualifier

NOTE: This API is typically used for user defined qualifier.

11.28.5 bcm_field_action_*

11.28.5.1 bcm_field_action_create()

```
bcm_field_action_create(
   int unit,
   uint32 flags,
   bcm_field_action_info_t * action_info,
   bcm_field_action_t * action_id);
```

Parameters:

- `unit` [IN]: device ID
- `flags` [IN]: `BCM_FIELD_WITH_ID` the action-ID should be explicitly indicated
- `action_info` [IN]: Structure holds detailed info of the qualifier to be created
  - `stage` – Stage that the qualifier will be defined in
  - `action_type` – Used as the basis for the user-defined action type, can be also void action
  - `size` – Size of the action to set in the payload in bits.
  - `prefix_size` – Number of prefix bits to be written (will not appear on the payload).
  - `prefix_value` – Value of the prefix (will be put in the MSB of the action).
  - `name` – String for application use.
- `action_id` [IN/OUT]: Handler-ID for the created qualifier. If the `BCM_FIELD_WITH_ID` flag is set is considered as input otherwise as output.

11.28.5.2 bcm_field_action_info_get()

```
bcm_field_action_info_get(
 int unit,
 bcm_field_action_t  action_id,
 bcm_field_stage_t stage,
 bcm_field_action_info_t * action_info);
```

Parameters:

- `unit` [IN]: Device ID
- `action_id` [IN]: Handler-ID for the action
- `stage` [IN]: The stage at which to get the action information. This parameter is relevant only for predefined actions.
- `action_info` [OUT]: See details of the struct in bcm_field_action_create().

11.28.5.3 bcm_field_action_destroy()

```
bcm_field_action_destroy(
 int unit,
 bcm_field_action_t  action_id);
```

Parameters:

- `unit` [IN]: Device ID
- `action_id` [IN]: Handler-ID for the action.

11.28.5.4 bcm_field_action_value_map(

```
bcm_field_action_value_map(
   int unit,
   bcm_field_stage_t stage,
   bcm_field_action_t  action_id,
   uint32 bcm_value[],
   uint32 hw_value[]);
```

Parameters:

- `unit` [IN]: Device ID
- `stage` [IN]: The stage the action is defined for
- `action_id` [IN]: handler-ID for the action to map the value for
- `bcm_value` [IN]: The BCM value to map
- `hw_value` [OUT]: The mapped HW value of the given BCM value for the given action

NOTE: This API is typically used for FEM and `FES_add`.

11.28.6 bcm_field_entry_*

11.28.6.1 bcm_field_entry_add()

```
bcm_field_entry_add(
   int unit,
   uint32 flags,
   bcm_field_group_t fg_id,
   bcm_field_entry_info_t * entry_info,
   bcm_field_entry_t*entry_handle)
```

Parameters:

- `unit` [IN]: Device ID
- `flags`[IN]: `BCM_FIELD_FLAG_WITH_ID`/`BCM_FIELD_FLAG_UPDATE`
  Supported for field group types TCAM Lookup, and EXEM.
- `fg_id`[IN]: Field group ID to add entries to [relevant for TCAM and EXEM FG]
- `entry_info` [IN]:
  - `priority` – The lower the number the higher the entry priority. A higher priority lookup will hit first. In field groups with `bcmFieldTcamBankAllocationModeSelectWithLocation` bank allocation mode, this parameter acts as the exact location of the entry to be added.

NOTE: The entry priority cannot be changed by using the `BCM_FIELD_FLAG_UPDATE` flag.

  - `nof_entry_quals` – Number of meaningful qualifiers in the key.
  - `entry_quals` – Array of qualifier info structure:
    - `type` – One of the qualifiers was set for the field group at field group add API
    - `value` – Set the value for the qualification (lookup key). For ranged qualifiers, this is the minimum value of the range to match.
    - `max_value` – For ranged qualifiers only. this is the maximal value of the range to match.
    - `mask` – Determines which of the value bits are relevant for lookup. Only relevant for TCAM (external or otherwise). It is not relevant for ranged qualifiers.
  - `nof_entry_actions` – Number of meaningful actions in payload
  - `entry_actions`
    - type – The action that was set for the field group at field group add API
    - value – Set value for the action (lookup result)
  - `core_id` – The core to install an entry on

NOTE: The entry `core_id` cannot be changed by using the `BCM_FIELD_FLAG_UPDATE` flag.

  - `valid_bit` – (Only for TCAM) Specifies whether this entry is valid in HW or not, in case entry is not valid, it will only act as a placeholder and will not affect traffic.
- `entry_handle_p` [IN/OUT]: Holds a unique ID for created entry (only for TCAM FGs). This ID can be used to delete or get the entry. INPUT when `BCM_FIELD_FLAG_WITH_ID` or `BCM_FIELD_FLAG_UPDATE` flags are set; OUTPUT otherwise.

11.28.6.1.1 Creating a Custom entry_handle

When creating a new custom `entry_handle` to use as INPUT while creating an entry `BCM_FIELD_FLAG_WITH_ID` flag set, make sure the requested `entry_handle` has the following correct format:

For TCAM:

- Bit 31:24 – Reserved
- Bit 23:20 – Core-ID
- Bit 19:0 – Access ID (Custom)

For TCAM:

```
entry_handle = core_id << 20 | access_id;
```

NOTE: `access_id` must be available for the entry addition to work. If the `access_id` is already in use by another entry, the API call will fail until the entry is deleted.

NOTE: External TCAM is not available for the BCM88830.

11.28.6.2 bcm_field_entry_info_get()

```
bcm_field_entry_info_get(
   int unit,
   bcm_field_group_t fg_id,
   bcm_field_entry_t entry_handle,
   bcm_field_entry_info_t * entry_info)
```

Parameters:

- `unit` [IN]: Device ID
- `fg_id` [IN]: Field group ID
- `entry_handle` [IN]: Handle-ID for entry
- `entry_info` [IN/OUT]: Structure as described in bcm_field_entry_add().
  - `entry_info.entry_quals` and `entry_info.entry_actions` include qualifiers or actions of type RAW only. For example, when a user has an entry with a qualifier of type `InPort`, the user provides a gport that gets translated into a port number. When the user gets this entry, the port number value is returned and the qualifier returned is `InPortRaw`.
- The rest of the structure variables remain OUTPUT variables.

NOTE:

- The RAW qualifier or action means the values is the same as in hardware. No encoding or mapping is performed.
- Each qualifier/action has a BCM name for a RAW qualifier or action
- This API will always return RAW qualifiers or actions.

11.28.6.3 bcm_field_entry_delete()

```
bcm_field_entry_delete(
   int unit,
   bcm_field_group_t fg_id,
   bcm_field_entry_qual_t * entry_qual_info,
   bcm_field_entry_t entry_handle)
```

Parameters:

- `unit` [IN]: Device ID
- `fg_id` [IN]: Field group ID
- `entry_qual_info` [IN]: Pointer to struct of Entry qual information. Used only to add an EXEM entry. For everything else, set it to NULL.
- `entry_handle` [IN]: Handle-ID for entry (only for TCAM FG)

11.28.6.4 bcm_field_entry_hit_get()

```
bcm_field_entry_hit_get(
   int unit,
   uin32 flags,
   bcm_field_entry_t entry_handle,
   uint8 *entry_hit_core_bmp)
```

Parameters:

- `unit` [IN]: Device ID.
- `flags` [IN]: TBD.
- `entry_handle` [IN]: Handle-ID for TCAM entry to return its hit indication bit.
- `entry_hit_core_bmp` [OUT]: 2-bit bitmap, where each bit indicates whether there was a hit for the given entry on the corresponding core. For example, if bit 0 is true, there was a hit for the given entry on core 0.

11.28.6.5 bcm_field_entry_hit_flush()

```
bcm_field_entry_hit_flush(
   int unit,
   uin32 flags,
   bcm_field_entry_t entry_handle)
```

Parameters:

- `unit` [IN]: Device ID.
- `flags` [IN]: `BCM_FIELD_ENTRY_HIT_FLUSH_ALL` – Clear hit indication bit for all TCAM entries.
- `entry_handle` [IN]: Handle-ID for TCAM entry to clear its hit indication bit, this value is ignored when `BCM_FIELD_ENTRY_HIT_FLUSH_ALL` is given.

NOTE: This API clears the hit indication bit for the given TCAM entry (it only works for Internal-TCAM FGs) if the `BCM_FIELD_ENTRY_HIT_FLUSH_ALL` flag is given. Then, the function clears the hit indication bits for all TCAM entries.

11.28.7 bcm_field_ace_*

11.28.7.1 bcm_field_ace_format_add()

```
bcm_field_ace_format_add(
   int unit,
   uint32 flags,
   bcm_field_ace_format_info_t * ace_format_info,
   bcm_field_ace_format_t * ace_format_id)
```

Parameters:

- `unit` [IN]: Device ID
- `flags`[IN]: Can be `BCM_FIELD_FLAG_WITH_ID` – Add an ACE Format with ID.
- `ace_format_info` [IN]: Structure describing ACE format info:
  - `nof_actions`: Number of valid actions in the ACE format
  - `action_types`: Array of action types included in the ACE format.
  - `action_with_valid_bit`: Array indicating for each action whether it used a valid bit or not. If an action does not use a valid bit, that action must appear on all entries added to the ACE format with `bcm_field_ace_entry_add()`.
- `ace_format_id`[INOUT]: Handle-ID for ACE-format.

11.28.7.2 bcm_field_ace_format_info_get ()

```
bcm_field_ace_format_info_get (
   int unit,
   bcm_field_ace_format_t ace_format_id,
   bcm_field_ace_format_info_t * ace_format_info)
```

Parameters:

- `unit` [IN]: Device ID
- `ace_format_id`[IN]: Handle-ID for ACE-format.
- `ace_format_info`[OUT]: See description above

11.28.7.3 bcm_field_ace_format_delete()

```
bcm_field_ace_format_delete(
   int unit,
   bcm_field_ace_format_t ace_format_id)
```

Parameters:

- `unit` [IN]: Device ID
- `ace_format_id` [IN]: Handle-ID for ACE-format.

11.28.7.4 bcm_field_ace_entry_add()

```
bcm_field_ace_entry_add(
   int unit,
   uint32 flags,
   bcm_field_ace_format_t ace_format_id,
   bcm_field_ace_entry_info_t * entry_info,
   uint32 *entry_handle)
```

Parameters:

- `unit` [IN]: Device ID
- `flags`[IN]: Can be `BCM_FIELD_FLAG_WITH_ID` to add an entry with ID or `BCM_FIELD_FLAG_UPDATE` to be updated in an existing entry.
- `ace_format_id`[IN]: Handle-ID for ACE-format.
- `ace_entry_info`[IN]: Structure describing ACE entry info:
  - `nof_actions` : Number of valid actions in the ACE format
  - `entry_actions`: Array of structures each describing entry action:
    - Type – Action type
    - Value – Action value to be set in the ACE format payload.
- `entry_handle` [IN/OUT]: Pointer to the handle ID for the added entry. If a `BCM_FIELD_FLAG_WITH_ID` flag is set, it is considered as input; otherwise, it is considered as output.

11.28.7.5 bcm_field_ace_entry_info_get()

```
bcm_field_ace_entry_info_get(
   int unit,
   bcm_field_ace_format_t ace_format_id,
   uint32  entry_handle,
   bcm_field_ace_entry_info_t * entry_info)
```

Parameters:

- `unit` [IN]: device id
- `ace_format_id`[IN]: Handle ID for ACE-format.
- `entry_handle`[IN] : Handle ID for the entry
- `ace_entry_info`[OUT]: Structure as described above

11.28.7.6 bcm_field_ace_entry_delete()

```
bcm_field_ace_entry_delete(
   int unit,
   bcm_field_ace_format_t ace_format_id,
   uint32  entry_handle)
```

Parameters:

- `unit` [IN]: Device ID
- `ace_format_id`[IN]: Handle ID for ACE-format.
- `entry_handle`[IN]: Handle ID for the entry

11.28.8 bcm_field_fem_action_*

11.28.8.1 bcm_field_fem_action_add()

```
bcm_field_fem_action_add(
   int unit,
   uint32 flags,
   bcm_field_group_t fg_id,
   bcm_field_action_priority_t action_priority,
   bcm_field_fem_action_info_t * fem_action_info )
```

Parameters:

- `unit` [IN]: Unit ID
- `flags` [IN]: Future use
- `fg_id` [IN]: The FG to which the FEM action is added.
- `action_priority` [IN]: Encoded position of the FEM ID to use for the action (essentially, this is the identifier of the FEM. Caller calculates this value using `BCM_FIELD_ACTION_POSITION` with `array_id` set to 1 or 3 and `position` set to 0-7)
- FEM action info [IN]: Structure define all FEM action information:
  - FEM Input
    - Payload Offset – Offset inside the FG-ID result buffer, in a resolution of 16 bits. The FEM input is 32 bits long. When the FG result buffer is longer than 32 bits, offset is required
    - Second FG ID – If the FG-ID is valid, override the main FG's result buffer with the 16 bits resulting from this specified TCAM-FG result.
  - Condition MSB: Select the MSB of a 4-bit chunk that will choose the extraction table ID. This chunk is taken from the 32 bits that are selected by the Payload Offset described previously. The chunk value is an index into the Conditions table.
  - Conditions table[16]: The index of the table is the 4 bits selected by Condition MSB, each one contains:
    - is action valid: If TRUE, the FEM will produce an action for the given 4-bit chunk
    - Extraction table ID Choose one of four action extraction tables that will produce an action
  - Extractions[4], each one contains:
    - Action Type: Action type (type of `bcm_field_action_t`)
    - Output bits[24]: Array of extraction commands. Each extracts 1 bit of the action value (for fem-id=0,1 only four entries are meaningful). Each one contains:
      - Source: Indicate whether the action output bit is forced or taken from the key (0-forced, 1-taken from the 32 input bits)
      - Key offset: Position of bit on the 32 input bits
      - Forced value: Either 1 or 0
  - Increment: Add a constant value to the result of the extraction

11.28.8.2 bcm_field_fem_action_info_get()

```
bcm_field_fem_action_info_get(
   int unit bcm_field_group_t fg_id,
   bcm_field_action_priority_t action_priority,
   bcm_field_fem_action_info_t * fem_action_info);
```

Parameters:

- `unit` [IN]: unit ID
- `fg_id` [IN]: The FG to which the FEM action was added.
- `action_priority` [IN]: Encoded position of the FEM ID to use for the action. Essentially, this is the identifier of the FEM. This value is calculated using `BCM_FIELD_ACTION_POSITION` with `array_id` set to 1 or 3 and `position` set to 0-7.
- FEM action info [OUT]: structure as described above.

11.28.8.3 bcm_field_fem_action_delete()

```
bcm_field_fem_action_delete(
   int unit,
   bcm_field_group_t fg_id,
   bcm_field_action_priority_t action_priority);
```

Parameters:

- `unit` [IN]: unit ID
- `fg_id` [IN]: The FG to which the FEM action was added.
- `action_priority` [IN]: Encoded position of the FEM ID to use for the action. Essentially, this is the identifier of the FEM. This value is calculated using `BCM_FIELD_ACTION_POSITION` with `array_id` set to 1 or 3 and `position` set to 0-7.

11.28.9 bcm_field_efes_action_*

11.28.9.1 bcm_field_efes_action_add()

```
bcm_field_efes_action_add(
   int unit,
   uint32 flags,
   bcm_field_group_t fg_id,
   bcm_field_context_t context_id,
   bcm_field_action_priority_t action_priority,
   bcm_field_efes_action_info_t * efes_action_info )
```

Parameters:

- `unit` [IN]: Unit ID
- `flags` [IN]: Future use
- fg_id [IN]: The FG to which the FEM action was added.
- `context_id` [IN]: The context attached to the FG to which the EFES action was added.
- `action_priority` [IN]: Encoded position of the EFES to use for the action. Essentially, this is the identifier of the added EFES (together with the field group and context). The caller calculates this value using `BCM_FIELD_ACTION_POSITION` with `array_id` set to 0 or 2 (the latter for iPMF1/2 only) and `position` set to 0 to 16 (or 0 to 12 in ePMF).

- EFES action info [IN]: Structure define all EFES action information:
  - `is_condition_valid` – Whether to use the EFES condition feature. If true and the field group type is direct extraction, `bcm_field_group_context_attach()` should have been called with the flag `BCM_FIELD_FLAG_32_RESULT_MSB_ALIGN`.
  - `condition_msb` – If `is_condition_valid` is true, this is the offset within the field group's payload of the 2-bit condition. Use `bcm_field_group_action_offset_get()` as a reference point for the offset.
  - `efes_condition_conf` – A four-element array of the configuration for each of the four options given by the two condition bits. if `is_condition_valid` is false, only fill the first element.
    - `is_action_valid` – If false, this option performs no action.
    - `action_type` – The action to be performed. Only use predefined actions.
    - `action_lsb` – The starting offset in the field group's payload to take data from for the action value. Use `bcm_field_group_action_offset_get()` as a reference point for the offset.
    - `action_nof_bits` – Number of bits to take from the payload.
    - `action_with_valid_bit` – Indicates if valid bit is to be used. if so, its offset would be (`action_lsb` – 1).
    - `valid_bit_polarity` – The polarity of the valid bit (the value the valid bit should have for the action to be performed) if `action_with_valid_bit` is true.
    - `action_or_mask` – A bitwise or that is performed on the action value.

11.28.9.2 bcm_field_efes_action_info_get()

```
bcm_field_efes_action_info_get(
   int unit,
   bcm_field_group_t fg_id,
   bcm_field_context_t context_id,
   bcm_field_action_priority_t action_priority,
   bcm_field_efes_action_info_t * efes_action_info);
```

Parameters:

- `unit` [IN]: Unit ID.
- `fg_id` [IN]: The field group to which the EFES was added.
- `context_id` [IN]: The context of the FG to which the EFES was added.
- `action_priority` [IN]: Encoded position of the EFES to use for the action. Essentially, this is the identifier of the added EFES (together with the field group and context). The caller calculates this value using `BCM_FIELD_ACTION_POSITION` with `array_id` set to 0 or 2 (the latter for iPMF1/2 only) and `position` set to 0 to 16 (or 0 to 12 in ePMF).
- EFES action info [OUT]: structure as described in the previous section.

11.28.10 bcm_field_compare_*

11.28.10.1 bcm_field_compare_operand_offset_get()

```
bcm_field_compare_operand_offset_get(
    int unit,
    int pair_id,
    bcm_field_compare_operand_t compare_operand,
    int *offset);
```

Parameters:

- `unit` [IN]: Unit ID
- `pair_id` [IN]: Compare pair-id
- `compare_operand` [IN]: The compare operand. It shows which comparison result should be retrieved. It can be one of the following:
  - equal
  - not equal
  - smaller
  - not smaller
  - bigger
  - not bigger.
- `offset` [Out]: The offset of the operand in the 6-bits result buffer.

11.28.11 bcm_field_entry_hit_*

11.28.11.1 bcm_field_entry_hit_get()

```
bcm_field_entry_hit_get(
    int unit,
    uint32 flags,
    bcm_field_entry_t entry_handle,
    uint8 *entry_hit_core_bmp);
```

Parameters:

- `unit` [IN]: Unit ID
- `flags` [IN]: Currently not in use (future support)
- `entry_handle` [IN]: Entry handle for which the hit indication information is retrieved.
- `entry_hit_core_bmp` [Out]: A 2-bit bitmap, where every bit indicates on which core the entry was hit: bit 0 = core 0, bit 1 = core 1. There is a possibility that the entry was hit on both cores. In this case, both bits are set.

11.28.11.2 bcm_field_entry_hit_flush()

```
bcm_field_entry_hit_flush(
    int unit,
    uint32 flags,
    bcm_field_entry_t entry_handle);
```

Parameters:

- `unit` [IN]: Unit ID
- `flags` [IN]: Flags to indicate what action should be performed by the API, currently in use is `BCM_FIELD_ENTRY_HIT_FLUSH_ALL` which is used for flushing hit indication info for all entries. No need to specify an entry ID. If no flag is specified, the API clears information only for the given `entry_handle`.
- `entry_handle` [IN]: Entry handle for which hit indication info is flushed.

11.28.12 bcm_field_tcam_*

11.28.12.1 bcm_field_tcam_bank_evacuate()

```
bcm_field_tcam_bank_evacuate(
    int unit,
    uint32 flags,
    bcm_field_tcam_bank_evacuate_info_t *evac_info);
```

Parameters:

- `unit` [IN]: unit ID
- `flags` [IN]: Reserved
- `evac_info` [IN]:
  - `fg_id` – Field group ID from which to evacuate banks (use `fg_id` or `pp_app`, but not both)
  - `pp_app` – PP-Application to evacuate banks from (use `fg_id` or `pp_app`, but not both)
  - `nof_banks` – Number of banks to evacuate
  - `tcam_bank_ids` – Bank IDs to evacuate (0 to 11 are big banks, and 11 to 15 are small banks)
  - `core_ids` – Array of core IDs for each bank ID (initialized to `BCM_CORE_ALL`)

11.28.12.2 bcm_field_tcam_bank_add()

```
bcm_field_tcam_bank_add(
    int unit,
    uint32 flags,
    bcm_field_tcam_bank_info_t * add_info)
```

Parameters:

- `unit` [IN]: Unit ID
- `flags` [IN]: Reserved
- `add_info` [IN]:
  - `fg_id` – Field group ID to add banks to (use `fg_id` or `pp_app`, but not both)
  - `pp_app` – PP-Application to add banks to (use `fg_id` or `pp_app`, but not both)
  - `nof_banks` – Number of banks to add
  - `tcam_bank_ids` – Bank IDs to add (0 to 11 are big banks, and 11 to 15 are small banks)
  - `core_ids` – Array of core IDs for each bank ID (initialized to `BCM_CORE_ALL`)

NOTE: This function allows adding (allocate) new banks to certain FG and PP applications.

11.28.12.3 bcm_field_tcam_bank_status_get

```
bcm_field_tcam_bank_status_get(
   int         unit,
   bcm_core_t  core,
   uint32      bank_id,
   bcm_field_tcam_bank_status_info_t * bank_status_info)
```

Parameters:

- `unit` [IN]: Unit ID
- `core` [IN]: Core ID to query bank status on
- `bank_id` [IN]: Bank ID to query information for
- `bank_status_info`[OUT]:
  - `fg_ids[]` – Array of Field group IDs that allocate the given bank terminated by `DNX_FIELD_GROUP_INVALID`
  - `pp_app[]` – Array of PP applications that allocate the given bank terminated by `bcmFieldAppDbInvalid`
  - `nof_fg_entries[]` – Array that represents the number of entries per field group on the given bank
  - `nof_pp_app_entries[]` – Array that represents the number of entries per PP application on the given bank
  - `nof_free_half_entries` – Number of free half-sized entries left on the bank
  - `nof_free_single_entries` – Number of free single-sized entries left on the bank
  - `nof_free_double_entries` – Number of free double-sized entries left on the bank

NOTE: A different bank status exists per core because TCAM entries can be installed per core.

11.28.13 bcm_field_range_*

11.28.13.1 bcm_field_range_set()

```
bcm_field_range_set(
    int unit,
    uint32 flags,
    bcm_field_stage_t stage,
    bcm_field_range_t range_id,
    bcm_field_range_info_t * range_info)
```

Parameters:

- `unit` [IN]: Unit ID
- `flags` [IN]: Reserved
- `stage` [IN]: one of supported field stages
- `range_d` [IN]: Range to configure
- `range_info` [IN]:
  - `range_type`:
    - `bcmFieldRangeTypeL4SrcPort` – L4 Source Port Range Qualifier. `bcmFieldQualifyL4PortRangeCheck` used as Key in FG (range ID is shared with L4Dst)
    - `bcmFieldRangeTypeL4DstPort` – L4 Destination Port Range Qualifier. `bcmFieldQualifyL4PortRangeCheck` used as key in FG (range ID is shared with L4Src)
    - `bcmFieldRangeTypeOutVport` – Egress Virtual Port Range Qualifier. `bcmFieldQualifyVPortRangeCheck` CS Qualifier
    - `bcmFieldRangeTypePacketHeaderSize` – Packet Header Size Range Qualifier. `bcmFieldQualifyPacketLengthRangeCheck` – Hold the first range ID hit

    The following fields are supported only in the BCM88830 device:
    - `bcmFieldRangeTypeL4OpsPacketHeaderSize` – Packet Header Size Range Qualifier for L4 OPS.
    - `bcmFieldRangeTypeInTTL` – In TTL Range Qualifier
    - `bcmFieldRangeTypeUserDefined1Low` – User-Defined-1 Low (16b) Range Qualifier
    - `bcmFieldRangeTypeUserDefined1High` – User-Defined-1 High (16b) Range Qualifier
    - `bcmFieldRangeTypeUserDefined2Low` – User-Defined-2 Low (16b) Range Qualifier

    - bcmFieldRangeTypeUserDefined2High – User-Defined-2 High (16b) Range Qualifier
  - min_val – Minimum value of range
  - max_val – Maximum number of range

11.28.13.2 bcm_field_range_type_config_set()

```
bcm_field_range_type_config_set(
    int unit,
    uint32 flags,
    bcm_field_stage_t stage,
    bcm_field_context_t context_id,
    uint32 configurable_range_index,
    bcm_field_range_qual_info_t *range_qualifier_info);
```

Parameters:

- unit [IN]: Unit ID
- flags [IN]: Reserved
- stage [IN]: Stage for which to configure the range FFC
- context_id [IN]: Reserved
- configurable_range_index [IN]: Index of the range to configure
- range_qualifier_info [IN]:
  - qual_type – The qualifier type with which to configure the FFC
  - qual_info – The same as bcm_field_group_context_attach()

11.28.13.3 bcm_field_range_type_config_get()

```
bcm_field_range_type_config_get(
    int unit,
    uint32 flags,
    bcm_field_stage_t stage,
    bcm_field_context_t context_id,
    uint32 configurable_range_index,
    bcm_field_range_qual_info_t * range_qualifier_info)
```

Parameters:

- unit [IN]: Unit ID
- flags [IN]: Reserved
- stage [IN]: Stage to configure the range FFC for
- context_id [IN]: Reserved
- configurable_range_index [IN]: Index of the range with the configuration to retrieve
- range_qualifier_info [OUT]: The same as in bcm_field_range_type_config_set()

11.28.14 bcm_field_name_*

11.28.14.1 bcm_field_name_to_id()

```
bcm_field_name_to_id(
    int unit,
    uint32 flags,
    bcm_field_name_to_id_info_t * name_to_id_info,
    int *nof_ids,
    uint32 ids[BCM_FIELD_NAME_TO_ID_MAX_IDS]);
```

Parameters:

- `unit` [IN]: Unit ID
- `flags` [IN]: Reserved
- `name_to_id_info` [IN]:
  - `stage` – The stage in which we look for the object. Only relevant for contexts.
  - `name_to_id_type` – What type of object we want to convert to ID.
  - `name` – The name of the object we want to find an ID for.
- `nof_ids` [OUT]: The number of objects in the array IDs.
- `ids` [OUT]: An array with all of the IDs matching the provided name.

11.29 Hashing Parameters

Table 5: CRC-16 Functions Enum

CRC16 Value	CRC16 Enum
0	bcmFieldContextHashFunctionFirstReserved
1	bcmFieldContextHashFunctionSecondReserved
2	bcmFieldContextHashFunctionThirdReserved
3	bcmFieldContextHashFunctionCrc16Bisync
4	bcmFieldContextHashFunctionCrc16Xor1
5	bcmFieldContextHashFunctionCrc16Xor2
6	bcmFieldContextHashFunctionCrc16Xor4
7	bcmFieldContextHashFunctionCrc16Xor8
8	bcmFieldContextHashFunctionXor16
9	bcmFieldContextHashFunctionCrc16Ccitt
10	bcmFieldContextHashFunctionCrc32ALow
11	bcmFieldContextHashFunctionCrc32Ahigh
12	bcmFieldContextHashFunctionCrc32BLow
13	bcmFieldContextHashFunctionCrc32BHigh

Table 6: Hash Function on Key

Hash Function	Description
bcmFieldContextHashActionValueNone	Key stays as-is
bcmFieldContextHashActionValueReplaceCrc	Key is replaced with CRC16
bcmFieldContextHashActionValueAugmentCrc	Key is augmented with CRC16
bcmFieldContextHashActionValueAugmentKey	Key is augmented with KeyY[63:0]

Table 7: Hash Action Key

Hash Action Key	Description
bcmFieldContextHashActionKeyEcmpLbKey0	Hashing key ECMP_LB0
bcmFieldContextHashActionKeyEcmpLbKey1	Hashing key ECMP_LB1
bcmFieldContextHashActionKeyEcmpLbKey2	Hashing key ECMP_LB2
bcmFieldContextHashActionKeyNetworkLbKey	Hashing key NETWORK_LB
bcmFieldContextHashActionKeyLagLbKey	Hashing key LAG_LB
bcmFieldContextHashActionKeyAdditionalLbKey	Hashing key ADDITIONAL_LB

11.30 Common Types

11.30.1 Common Enum Supported Values

Table 8: Supported Layer Type Values
(Values for bcmFieldQualifyForwardingType, bcmFieldQualifyLayerRecordType, and bcmFieldActionParsingStartType)

Value	Description	Relevant for bcmFieldActionParsingStartType
bcmFieldLayerTypeEth	Ethernet	✓
bcmFieldLayerTypeIp4	Ipv4	✓
bcmFieldLayerTypeIp6	Ipv6	✓
bcmFieldLayerTypeMpls	MPLS (Tunnel)	✓
bcmFieldLayerTypeArp	ARP	✓
bcmFieldLayerTypeFcoe	Slow Control over Ethernet	✓
bcmFieldLayerTypeTcp	TCP	—
bcmFieldLayerTypeUdp	UDP	—
bcmFieldLayerTypeBfdSingleHop	BFD Single Hop	—
bcmFieldLayerTypeBfdMultiHop	BFD Multiple Hop	—
bcmFieldLayerTypeY1731	Y.1731	—
bcmFieldLayerTypeIcmp	ICMP	—
bcmFieldLayerTypeBierTi	Bier	—
bcmFieldLayerTypeBierMpls	Bier MPLS	—
bcmFieldLayerTypeRch	RCH	—
bcmFieldLayerTypePppoe	PPP over Ethernet	✓
bcmFieldLayerTypeSrv6Endpoint	SRv6 EndPoint	✓
bcmFieldLayerTypeSrv6Beyond	SRv6 Beyond	✓
bcmFieldLayerTypeIgmp	IGMP	—
bcmFieldLayerTypeIpAny	Any IP	—
bcmFieldLayerTypeIpt	IPT	—
bcmFieldLayerTypeTm	Traffic Management	✓
bcmFieldLayerTypeUnknown	Unknown	✓
bcmFieldLayerTypeSctp	SCTP	✓
bcmFieldLayerTypeForwardingMPLS	MPLS (Forwarding)	✓
bcmFieldLayerTypePtpGeneral	PTP message and PTP event	—
bcmFieldLayerTypeGtp	GTP	—
bcmFieldLayerTypeEthEnd	Used in SRv6. Indicates the parser should set `layer_protocol` to Ethernet and stop parsing the packet. Can be used only as Parsing Start type	✓
bcmFieldLayerTypeMplsEnd	Used in SRv6. Indicates the parser should set `layer_protocol` to MPLS and stop parsing the packet. Can be used only as a Parsing Start type.	✓
bcmFieldLayerTypeSrv6EndpointPsp	SRv6 Endpoint PSP. Can be used only as a Parsing Start type.	✓
bcmFieldLayerTypeSrv6UspExtPsp	SRv6 RCH USP, PSP, and PSP extended Can be used only as a Parsing Start type	✓

In the following table, the Qualifier for tunnel types is `bcmFieldQualifyIpTunnelType`. If a tunnel is not supported by the table, `bcmFieldQualifyIpTunnelTypeRaw` can be used.

Table 9: Tunnel Types bcm_field_TunnelType_t

Value	Comments
bcmFieldTunnelTypeGre4	—
bcmFieldTunnelTypeGre8WithKey	—
bcmFieldTunnelTypeGre12	—
bcmFieldTunnelTypeUdp	—
bcmFieldTunnelTypeVxlan	Over UDP
bcmFieldTunnelTypeGpe	Over UDP
bcmFieldTunnelTypeGeneve	Over UDP
bcmFieldTunnelTypeL2TPv3OverUdp	—

Qualifiers for the IPv6 additional header are bcmFieldQualifyIp6FirstAdditionalHeader and bcmFieldQualifyIp6SecondAdditionalHeader.

Table 10: IPv6 Additional Header bcm_field_ip6_additional_header_type_t

Value	Comments
bcmFieldIp6AdditionalHeaderNotAvailable	No additional header
bcmFieldIp6AdditionalHeaderGre4	Tunnel over IP
bcmFieldIp6AdditionalHeaderGre8	Tunnel over IP
bcmFieldIp6AdditionalHeaderGre12	Tunnel over IP
bcmFieldIp6AdditionalHeaderUdp	—
bcmFieldIp6AdditionalHeaderL2tpv3	Tunnel over UDP
bcmFieldIp6AdditionalHeaderVxlan	Tunnel over UDP
bcmFieldIp6AdditionalHeaderVxlanGpe	Tunnel over UDP
bcmFieldIp6AdditionalHeaderSrv6	—
bcmFieldIp6AdditionalHeaderSrv6Sl0	—
bcmFieldIp6AdditionalHeaderExtensionHopByHop	—
bcmFieldIp6AdditionalHeaderExtensionFragment	—
bcmFieldIp6AdditionalHeaderExtensionAuth	—
bcmFieldIp6AdditionalHeaderExtensionDestination	—
bcmFieldIp6AdditionalHeaderExtensionMobility	—
bcmFieldIp6AdditionalHeaderExtensionHostProtocol	—
bcmFieldIp6AdditionalHeaderExtensionShim6	—

The qualifier for formats is bcmFieldQualifyL2Format.

Table 11: L2 Format bcm_field_L2Format_t

Value	Comments
bcmFieldL2FormatEthII	EtherType is above 1500
bcmFieldL2FormatSnap	Sub-Network Access Protocol
bcmFieldL2FormatLlc	Logical Link Control
bcmFieldL2FormatAny	Other type

The ePMF qualifier for forward context is bcmFieldQualifyContextId.

Table 12: Forward Context bcm_field_forward_context_t

Value	Description
bcmFieldForwardContextEth	Ethernet
bcmFieldForwardContextMirrorOrSs	—
bcmFieldForwardContextIPv4UcR0	IPv4 UC R0
bcmFieldForwardContextIPv4McR0	IPv4 MC R0
bcmFieldForwardContextIPv6UcR0	IPv6 UC R0
bcmFieldForwardContextIPv6McR0	IPv6 MC R0
bcmFieldForwardContextBierMpls	BIER MPLS BFR
bcmFieldForwardContextBierTi	BIER non-MPLS BFR
bcmFieldForwardContextIngressTrapLegacy	—
bcmFieldForwardContextCpuPort	—
bcmFieldForwardContextRchEnc	—
bcmFieldForwardContextRchPtchEnc	—
bcmFieldForwardContextStackingPort	—
bcmFieldForwardContextSrv6Endpoint	SRv6 endpoint
bcmFieldForwardContextSrv6EndpointPsp	SRv6 endpoint PSP
bcmFieldForwardContextTm	Traffic management
bcmFieldForwardContextRawProcessing	—
bcmFieldForwardContextTdm	—
bcmFieldForwardContextMplsInjectedFromOamp	—
bcmFieldForwardContextErppTrap	—
bcmFieldForwardContextDoNotEdit	—
bcmFieldForwardContextMpls	MPLS
bcmFieldForwardContextIpv4UcR1	IPv4 UC R1
bcmFieldForwardContextIpv4McR1	IPv4 MC R1
bcmFieldForwardContextIpv6UcR1	IPv6 UC R1
bcmFieldForwardContextIpv6McR1	IPv6 MC R1

Mapping of the TCAM PP applications for usage in TCAM advanced APIs

Table 13: TCAM PP Apps

Value	Reference
bcmFieldAppDbVlanPort	See Chapter 21, Ethernet Bridge
bcmFieldAppDbIpmcV4	See Section 22.14, Adding IPMC Entries
bcmFieldAppDbIpmcV6	See Section 22.14, Adding IPMC Entries
bcmFieldAppDbTunnelTerminatorV4	See Chapter 28, IP Tunnel v4 Termination
bcmFieldAppDbTunnelTerminatorBudV4	See Chapter 28, IP Tunnel v4 Termination
bcmFieldAppDbTunnelTerminatorV6	See Chapter 30, IP Tunnel v6 Termination
bcmFieldAppDbTunnelTerminatorBudV6	See Chapter 30, IP Tunnel v6 Termination
bcmFieldAppDbMplsFrr	See Section 24.5, MPLS Fast Reroute (FRR).
bcmFieldAppDbOamIdentification	See Section 33.24, Signal Degrade Indication
bcmFieldAppDbMplsIpVXParse	See Section 3, L2 and L3 Ports (GPORTs).

11.30.2 Layer Record Structure

11.30.2.1 General Structure

The layer record field is structured from:

- bcmFieldQualifyLayerRecordOffset – 8b offset of the layer from start of the packet (LSB)
  - offset is in bytes resolution
- bcmFieldQualifyLayerRecordQualifier – 16b qualifier
  - For supported list per device
    - See the Layer Record Qualifiers List table for the device.
    - Run the diagnostic `field qual pre bcm class=layer`
- bcmFieldQualifyLayerRecordType – 5b protocol type (MSB)
  - For the supported list, see Table 9.

Layer record build {MSB, LSB}:

{bcmFieldQualifyLayerRecordType, bcmFieldQualifyLayerRecordQualifier , bcmFieldQualifyLayerRecordOffset}

In context selection, the following qualifiers are equivalent (but the index of the layer is relative to the forwarding layer):

- `bcmFieldQualifyForwardingType` to `bcmFieldQualifyLayerRecordType`.
- `bcmFieldQualifyForwardingLayerQualifier` to `bcmFieldQualifyLayerRecordQualifier`.
- `offset` is not supported in context selection.

11.30.3 Layer Records Qualifier Structure

11.30.3.1 Get the Structure by SDK Code

The following configuration example gets the offset of a record qualifier for a specific layer (inside a specific layer record). In this example, its IpFrag qualifier is relevant only for the IPv4 layer and indicates whether the packet is fragmented:

```
bcm_field_qualifier_info_get_t qual_info;
int unit = 0;
bcm_field_stage_t stage=  bcmFieldStageIngressPMF1; //For LR stage is not relevant
bcm_field_qualify_t LR_qual = bcmFieldQualifyIpFrag;
bcm_field_qualifier_info_get(unit,LR_qual,stage,qual_info);
print qual_info;
bcm_field_qualifier_info_get_t qual_info = {
    bcm_field_qualifier_class_t qual_class = bcmFieldQualifierClassLayerRecord (2)
    int size = 1 (0x8)
    int offset = 10 (0x0)
```

The offset is inside whole layer records structure as explained above, so to get the offset of specific qualifier inside the `bcmFieldQualifyLayerRecordQualifier` field need to subtract `bcmFieldQualifyLayerRecordOffset` size (by using the same API, the size is 8).

ATTENTION: It is the user's responsibility to overlook (skip) the layer record data when the offset of the next layer is larger than 144 bytes. This is because only a buffer of 144B of the packet is available to the parser, so the parser output for data above that value is undefined.

11.30.3.2 UDP Parsing over IPv4 and IPv6 Layers

There are two options in both ingress and egress parser for UDP parsing:

- Collapsed UDP – Where the UDP header is parsed as part of the IP header.
- UnCollapsed UDP – Where the UDP is parsed as an extra header, after the IP header.

The Collapsed UDP header is available only for a specific set of `UDP.Dest_port` values, and also depends by a configuration that enables it per destination port value.

The APIs to enable the collapsed UDP are:

- `bcm_switch_control_indexed_set(unit, key, value)` – With key as: `bcmSwitchL3TunnelL2TP`, bcmSwitchL3TunnelVxLan, `bcmSwitchL3TunnelGtpU`.
- `bcm_switch_control_set(unit, type, arg)` with the following types:
  - `bcmSwitchUdpTunnelIPv4DstPort`
  - `bcmSwitchUdpTunnelIPv6DstPort`
  - `bcmSwitchUdpTunnelMplsDstPort`
  - `bcmSwitchVxlanUdpDestPortSet`
  - `bcmSwitchVxlanGpeUdpDestPortSet`

11.31 Information Specific to the BCM88690 (Jericho2) Device

The following table shows the number of various resources per stage:

Resource	Stage			
	iPMF-1	**iPMF-2**	**iPMF-3**	**ePMF**
context Selection lines	128	128	128	128
contexts	64		64	64
Total Number of keys	5+ 5 initial keys	5	3	3
TCAM lookup keys	4	4	2	2
Direct Extraction keys	0	2	1	0
Exact Match Keys	1	1		1
FES-Contexts	32		16	12
FEM	16		0	0

Table 14: BCM88690 Database Properties

Type	TCAM Lookup
Maximum key size	Possible key sizes
	80b/160b/320b
Maximum payload size	32b/64b/128b
Memory	Shared (with other TCAM FGs)
Maximum number of entries per FG	50K/25K/12.5K
Supported stages	iPMF1/2/3 ePMF
Maximum FGs (in a given stage) per Context	4/4/2 – iPMF1/2 Double key allowed combinations: A+B, C+D, F+G, H+I 2/2/1 – iPMF3 2/2/1 – ePMF **NOTE:** For ePMF, double lookup key size must be > 160b.

11.31.1 CS Qualifiers

To see the latest supported context selection qualifiers, use the `field map` diag. To see the latest version support, run diag: `field qualifier ContextSelection`.

Table 15: List of iPMF1 Stage Context Selection Qualifiers

bcmFieldQualify…	Description
ForwardingType	Use this qualifier together with `input_arg` to indicate which layer number relative to forwarding layer to qualify upon. Possible layer types are `bcmFieldLayerTypeXXX`.

Table 15: List of iPMF1 Stage Context Selection Qualifiers (Continued)

bcmFieldQualify…	Description
VlanFormat	Incoming VLAN structure format (untagged/single/double/priority-tagged). For possible values, refer to bcm_port_tag_struct_type_t.
RxTrapCode	Qualify on RX trap code.
Ptch	Opaque attribute field of the Injected packets. Part of the `PTCH_2` header. The input expects a raw value.
VPortRangeCheck	Virtual port range check results.
AppType	Packet Application type used by the ACL lookup. Can be either predefined (`bcm_field_AppType_t`) or user defined.
ForwardingLayerIndex	Qualify on the forwarding layer index.
ForwardingLayerQualifier	Use this qualifier together with `input_arg` (range [–1, +2]) to indicate which layer number relative to forwarding layer to qualify upon. It qualifies Layer Qualifiers (look for layer record qualifiers in UM).
TracePacket	Qualifies upon packets that trace was set for them.
InVportClass	Qualify on the profile of an in LIF encoded as gport.
PortClassPacketProcessing	PP port profile.
PortClassTrafficManagement	TM port profile.
AppTypePredefined	The predefined AppType (`bcm_field_AppType_t`) used by the forwarding lookup.
VlanFormatRaw	Qualifies on the incoming VLAN structure format (untagged, single tagged, double-tagged, or priority-tagged). The input expects a raw value.
AppTypeRaw	Qualifies on the packet application type used by the ACL lookup. Can be either predefined or user-defined. The input expects a raw value.
AppTypePredefinedRaw	Qualifies on the predefined AppType used by the forwarding lookup. The input expects a raw value.
ForwardingTypeRaw	The same qualifier as bcmFieldQualifyForwardingType but does not have a mapping function. In other words, the input expects a raw value (HW value).
PrtQualifier	Qualifies on the prt_qualifier signal, which may be set by the injected packet (Opaque attribute field part of the PTCH_2 header). The input expects an enum value `bcm_field_prt_qualifier_t`. Similar to the ptch qualifier, which uses a raw value and not an enum.
AcInLifWideData	Qualifies on the Ac InLIF wide (generic) data.

Table 16: List of iPMF2 Stage Context Selection Qualifiers

bcmFieldQualify…	Description
CascadedKeyValue	Value of key cascaded from prior group in cascade
CompareKeysResult0	Compare keys result 0
CompareKeysResult1	Compare keys result 1

Table 17: List of iPMF3 Stage Context Selection Qualifiers

bcmFieldQualify…	Description
ForwardingType	Qualify upon Forwarding Layer Type. Possible layer types are bcmFieldLayerTypeXXX.
VPortRangeCheckt	Virtual port range check results
ContextId	Qualifies on the context ID
ForwardingLayerQualifier	Forwarding Layer qualifier

Table 17: List of iPMF3 Stage Context Selection Qualifiers (Continued)

bcmFieldQualify…	Description
RpfEcmpMode	RPF ECMP mode
PortClassPacketProcessing	PP port profile
PortClassTrafficManagement	TM port profile
ForwardingTypeRaw	The same qualifier as bcmFieldQualifyForwardingType but without the mapping function. That is, the input expects a raw value (HW value).

Table 18: List of Egress-PMF Stage Context Selection Qualifiers

bcmFieldQualify…	Description
SrcClassField	PPH value 2
DstClassField	PPH value 1
ForwardingType	Use this qualifier together with `input_arg` to indicate which layer number relative to forwarding layer to qualify upon. Possible layer types are bcmFieldLayerTypeXXX.
VlanFormat	Incoming VLAN structure format (untagged/single/double/priority-tagged). For possible values, see bcm_port_tag_struct_type_t.
FheiSize	DNX FHEI header size, in bytes.
PphPresent	Qualifies on PPH present field in ITMH.
UDHBase0	User-defined header 0 encoded type and size – value 1: type PMF size 32b.
UDHBase1	User-defined header 1 encoded type and size – value 1: type PMF size 32b.
UDHBase2	User-defined header 2 encoded type and size – value 1: type PMF size 32b.
UDHBase3	User-defined header 3 encoded type and size – value 1: type PMF size 32b.
ContextId	Qualifies on the context ID (see bcmFieldForwardContextXXX).
ForwardingLayerQualifier	Use this qualifier together with `input_arg` to indicate which layer number relative to forwarding layer to qualify upon. It qualifies Layer Qualifiers (look for layer record qualifiers in UM).
PortClassPacketProcessing	PP port profile.
InVportClass0	InLIF profile 0, by Gport.
VlanFormatRaw	Qualifies on the incoming VLAN structure format (untagged, single-tagged, double-tagged, or priority-tagged). The input expects a raw value.
LearnExtensionPresent	Qualifies on whether a PPH learn extension is present.
ForwardingTypeRaw	The same qualifier as bcmFieldQualifyForwardingType but without the mapping function. That is, the input expects a raw value (HW value).
ContextIdRaw	Qualifies on the forwarding context value. The input expects a raw value. Relevant for egress stage only.
FtmhAsePresent	Qualify packet on FTMH ASE present field.
FtmhAseType	Qualify packet on FTMH ASE type value.

11.31.2 Qualifiers List

To see the latest version support, run diag: `field qualifier Predefined bcm stage=iPMF1/2/3/ePMF class=meta/layer/header`.

11.31.2.1 Header Qualifiers List

Table 19: List of Header Qualifiers (Exists in iPMF1, iPMF2, Egress-PMF Stages)

bcmFieldQualify…	Description
SrcIp6	IPv6 source address
DstIp6	IPv6 destination address
SrcIp6High	Source IPv6 address (high/upper 64 bits, 64 MSB)
DstIp6High	Destination IPv6 address (high/upper 64 bits, 64 MSB)
SrcIp6Low	Source IPv6 address (low/lower 64 bits, 64 LSB)
DstIp6Low	Destination IPv6 address (low/lower 64 bits, 64 LSB)
SrcMac	Source MAC address
DstMac	Destination MAC address
SrcIp	Packet source IP address
DstIp	Packet destination IP address
Ip6FlowLabel	IPv6 flow label
L4SrcPort	L4 source port
L4DstPort	L4 destination port
Ip6NextHeader	Next protocol field in IPv6 header
Ip6TrafficClass	Traffic class field in IPv6 header
Ip6HopLimit	Hop count field in IPv6 header
TcpControl	TCP control flags
IpFlags	IP flags field
ExtensionHeaderType	Qualifies on next header field in first extension header
ArpSenderIp4	Sender IPv4 field of ARP header
ArpTargetIp4	Target IPv4 field of ARP header
ArpOpcode	Opcode field of ARP header.
Ip4Protocol	Qualify upon IPv4 protocol
Ip4Tos	Qualify upon IPv4 TOS
Ip4Ttl	Qualify upon IPv4 TTL
EtherTypeUntagged	802.1 Ethernet type, only for untagged frames
Tpid	VLAN TPID, offset of the value is inside the VLAN tag. Need to add the offset where the tag start in the Layer (inner/outer...)
VlanId	VLAN ID, offset of the value is inside the VLAN tag. Need to add the offset where the tag start in the Layer (inner/outer...)
VlanPri	VLAN P-bits, offset of the value is inside the VLAN tag. Need to add the offset where the tag start in the Layer (inner/outer...)
VlanCfi	VLAN CFI, offset of the value is inside the VLAN tag. Need to add the offset where the tag start in the Layer (inner/outer...)
VlanPriCfi	VLAN P-bits and CFI, offset of the value is inside the VLAN tag. Need to add the offset where the tag start in the Layer (inner/outer...)
MplsLabel	MPLS label
MplsLabelId	ID field of MPLS label

Table 19: List of Header Qualifiers (Exists in iPMF1, iPMF2, Egress-PMF Stages) (Continued)

bcmFieldQualify…	Description
MplsLabelTtl	TTL field of MPLS label
MplsLabelBos	BOS field of MPLS label
MplsLabelExp	TTL field of MPLS label

11.31.2.2 Layer Record Qualifiers List

Qualifier	Description
IpFrag	IP fragments
LayerRecordType	Qualifies on the Layer Record Type. For possible values look at `bcm_field_layer_type_t`
LayerRecordOffset	Qualifies on the Layer Record Offset
LayerRecordQualifier	Qualifies on the Layer Record qualifier
EthernetBroadcast	Qualify on Ethernet packets which destination MAC address is broadcast (MAC-DA[47:0] is all 1)
EthernetFirstTpidExist	Qualify on Ethernet packets for which their first VLAN TPID exists
EthernetFirstTpidIndex	Qualify on Ethernet packets first VLAN TPID Index
EthernetSecondTpidExist	Qualify on Ethernet packets for which their second VLAN TPID exists
EthernetSecondTpidIndex	Qualify on Ethernet packets second VLAN TPID Index
EthernetThirdTpidExist	Qualify on Ethernet packets for which their third VLAN TPID exists
EthernetThirdTpidIndex	Qualify on Ethernet packets third VLAN TPID Index
L2Format	Qualify on the type of the frame (see bcmFieldL2FormatXXX)
IpHasOptions	Qualify on IPv4 packets where the IP Header include options
IpFirstFrag	Qualify on IPv4 packets where the IP Header is fragmented and this is the first fragment
IpTunnelType	Qualify on IPv4 TunnelType (see bcmFieldTunnelXXX)
Ip6MulticastCompatible	Qualify on IPv6 packets for which their DIP is multicast (8 MSBs are 0xFF)
Ip6FirstAdditionalHeaderExist	Qualify on IPv6 packets' first additional header exists
Ip6FirstAdditionalHeader	Qualify on IPv6 packets' first additional header (see bcmFieldIp6AdditionalHeaderXXX)
Ip6SecondAdditionalHeaderExist	Qualify on IPv6 packets' second additional header exists
Ip6SecondAdditionalHeader	Qualify on IPv6 packets' second additional header (see bcmFieldIp6AdditionalHeaderXXX)
ItmhPphType	Qualify on Ingress TM Header PPH type
LayerRecordTypeRaw	Qualifies on the Layer Record Type. For possible values, look at bcm_field_layer_type_t; input expects raw value
Ip4DstMulticast	Qualify on IPv4 packets for which their DIP is multicast
IpTunnelTypeRaw	Raw Qualify on IPv4 TunnelType

11.31.2.3 Metadata Qualifiers List

Table 20: List of iPMF-1,2 Qualifiers

bcmFieldQualify…	Description
InPort	Ingress port. The value should be encoded as LOCAL GPORT type. Includes core ID, and can be used by entries for all cores.
L4PortRangeCheck	Qualify on L4 Ops (L4 source and dest ports) range. The qualifier is a bitmap and not specific range id, it means if range 2 was configured and the qualifier is hit, the entry value is expected to be 0x100
SrcPort	Qualify on source port, value should be encoded as SYS_PORT GPORT Type

Table 20: List of iPMF-1,2 Qualifiers (Continued)

bcmFieldQualify…	Description
DstPort	Qualify on destination port, all ingress stages expect raw value, egress expects GPORT encoded as SYS_PORT
InterfaceClassL2	VSI Profile
Vrf	VRF Id
VlanFormat	Incoming vlan structure format (untagged/single/double/priority-tagged). For possible values, see bcm_port_tag_struct_type_t.
DstMulticastGroup	Multicast Group id, encoded as GPORT
SrcModPortGport	Qualify on Source module/port pair, value should be encoded as SYS_PORT GPORT Type
Color	Qualify based on packet color, value should be one of BCM_FIELD_COLOR_XXX
IntPriority	Internal priority (TC)
Vpn	VSI Id
MplsForwardingLabelAction	MPLS forwarding label action
L2Learn	L2 learn enable
EcnValue	Qualify on metadata ECN / congestion information (including CNI)
Ptch	Opaque attribute field of the Injected packets. Part of the PTCH_2 header. The input expects a raw value.
RxTrapCode	Qualify on RX Trap Code
RxTrapData	Qualify on RX Trap qualifier
TrillEgressRbridge	Egress RBridge Nickname
ISid	I-SID (MAC-in-MAC lookup-id)
DstRpfGport	RPF destination (gport) for the RPF check
TrunkHashResult	Trunk Hash Result (that is, the Load-balancing Key)
PacketLengthRangeCheck	Packet length range, range ID on which the packet matched. If packet size > 144B, it is treated as 144B.
LearnSrcMac	Qualifies on learn source MAC
LearnVlan	Qualifies on learn VLAN
PacketSize	Packet size in Bytes
TranslatedOuterVlanId	Translated Outer VLAN ID
TranslatedOuterVlanPri	Translated Outer VLAN priority
TranslatedOuterVlanCfi	Translated Outer VLAN CFI
TranslatedInnerVlanId	Translated inner VLAN Id
TranslatedInnerVlanPri	Translated inner VLAN priority
TranslatedInnerVlanCfi	Translated inner VLAN CFI
RxTrapCodeForSnoop	Qualify on Snoop Code
IncomingIpIfClass	RIF profile
RxTrapStrength	Qualify on RX Trap Strength
OamUpMep	It indicates if the OAM packet is UP-MEP (sent to a destination in the network, as opposed to a specific port)
OamSubtype	In OAM the packet type is specified in the OAM header and mapped to a subtype in the hardware
OamHeaderOffset	indicates the offset of the OAM header relative to the start of packet (as opposed to start of header-offset)
OamStampOffset	indicates the offset to the position, in the OAM header, where the ToD or counter value should be stamped relative to the start of packet
OamMepId	If MEP is handled in OAMP, then the OAM-ID is the MEP-ID (equivalent to the index used to access the MEP DB)

Table 20: List of iPMF-1,2 Qualifiers (Continued)

bcmFieldQualify…	Description
OamMeterDisable	attribute that is passed to the PMF and can also be configured by the user per MEP
VlanAction	VLAN translation action ID
GeneratedTtl	Qualify on Packet's generate Time-To-Live
IpMulticastCompatible	Packet is compatible for multicast
PacketIsIEEE1588	Indicating whether the packet is 1588
IEEE1588Encapsulation	IEEE-1588 Encapsulation according to bcm_field_IEEE1588Encap_t
IEEE1588CompensateTimeStamp	IEEE-1588 update time stamp
IEEE1588Command	Command used by egress pipeline, indicating if CF (correction field) needs to be update
IEEE1588HeaderOffset	This field indicates the offset in bytes of the IEEE-1588 header
KeyGenVar	Match on configured key Gen Variable (values that was configured for context)
NetworkQos	Qualify Upon Network Qos
EcmpLoadBalanceKey0	ECMP Load Balance Key 0
EcmpLoadBalanceKey1	ECMP Load Balance Key 1
EcmpLoadBalanceKey2	ECMP Load Balance Key 2
ForwardingLayerIndex	Qualify on the Forwarding Layer Index
AcInLifWideData	Qualifies on the Ac InLIF wide (generic) data
NativeAcInLifWideData	Qualifies on the native Ac InLIF wide (generic) data
TracePacket	Qualifies upon Packets that Trace was set for them
InInterface	Ingress Logical Interface (LIF/RIF), value should be encoded as L3 interface
OutInterface	Egress Logical Interface (LIF/RIF), should be encoded as L3 interface
PortClassPacketProcessing	PP port profile
PortClassTrafficManagement	TM port profile
InVPort0	In LIF 0, value should be GPORT with subtype LIF
InVPort1	In LIF 1, value should be GPORT with subtype LIF
OutVPort0	OutLIF 0, value should be GPORT with subtype LIF
OutVPort1	OutLIF 1, value should be GPORT with subtype LIF
BierStringOffset	Qualifies upon Bier string offset
BierStringSize	Qualifies upon Bier string size
PacketIsBier	Qualifies upon Bier packets
PortClassPacketProcessing GeneralData	PP port general data
ForwardingAdditionalInfo	Qualifies on the Forwarding additional info
InVportClass0	InLIF Profile 0, by Gport
InVportClass1	InLIF Profile 1, by Gport
StatId0	Qualifies on statistics ID 0
StatId1	Qualifies on statistics ID 1
StatId2	Qualifies on statistics ID 2
StatId3	Qualifies on statistics ID 3
StatId4	Qualifies on statistics ID 4
StatId5	Qualifies on statistics ID 5
StatId6	Qualifies on statistics ID 6
StatId7	Qualifies on statistics ID 7
StatId8	Qualifies on statistics ID 8

Table 20: List of iPMF-1,2 Qualifiers (Continued)

bcmFieldQualify…	Description
StatId9	Qualifies on statistics ID 9
StatProfile0	Qualifies on statistics profile 0
StatProfile1	Qualifies on statistics profile 1
StatProfile2	Qualifies on statistics profile 2
StatProfile3	Qualifies on statistics profile 3
StatProfile4	Qualifies on statistics profile 4
StatProfile5	Qualifies on statistics profile 5
StatProfile6	Qualifies on statistics profile 6
StatProfile7	Qualifies on statistics profile 7
StatProfile8	Qualifies on statistics profile 8
StatProfile9	Qualifies on statistics profile 9
StatMetaData	Qualifies on Statistics metadata
InVPort0Raw	Qualifies on In LIF 0 input expects raw value
InVPort1Raw	Qualifies on In LIF 1 input expects raw value
OutVPort0Raw	Qualifies on OutLIF 0 input expects raw value
OutVPort1Raw	Qualifies on OutLIF 1 input expects raw value
MirrorCode	Matches on Mirror Code
LearnVsi	Qualifies on learn VSI
LearnData	Qualifies on data field in the Learn Info
LearnStationMove	Qualifies on learn Station move
LearnMatch	Qualifies on learn Match
LearnFound	Qualifies on learn Found
LearnExpectedWon	Qualifies on learn Expected Won
VrfValue	Qualifies on the VRF part of forwarding data, without checking that the type is VRF
InPortWithoutCore	Ingress Port, value should be encoded as LOCAL GPORT Type. Does not include the core ID, caution should be taken when not adding entry by core
SaLookupAcceptedStrength	Qualifies on Accepted age profile returned from SA Lookup result. Qualifier is only relevant for bridged packets
SrcPortRaw	Qualifies on source port, input expects raw value
InPortWithoutCoreRaw	Qualifies on Ingress Port. Does not include the core ID, caution should be taken when not adding entry by core, input expects raw value
VlanFormatRaw	Qualifies on Incoming VLAN structure format (untagged/single/double/priority-tagged), input expects raw value
InPortRaw	Qualifies on Ingress Port. Includes core ID, and can be used by entries for all cores, input expects raw value
ColorRaw	Qualifies on packet color, input expects raw value
DstRpfValid	Checks if the RPF Destination is valid
RpfOutVPort	Qualifies on the RPF OUT LIF. Value should be GPORT with subtype LIF
RpfOutInterface	Qualifies on the RPF OUT LIF. Value should be L3 interface
RpfOutVPortRaw	Qualifies on the RPF OUT LIF
RpfRouteValid	Qualifies on the RPF Route valid
VipValid	Qualifies on SLLB Virtual Wire VW_VIP_VALID
VipId	Qualifies on SLLB Virtual Wire VW_VIP_ID
VIPMemberReference	Qualifies on SLLB Virtual Wire VW_MEMBER_REFERENCE

Table 20: List of iPMF-1,2 Qualifiers (Continued)

bcmFieldQualify…	Description
PccHit	Qualifies on SLLB Virtual Wire VW_PCC_HIT
PrtQualifier	Qualifies on the prt_qualifier signal, which may be set by the injected packet (Opaque attribute field part of the PTCH_2 header). The input expects an enum value `bcm_field_prt_qualifier_t`. Similar to the ptch qualifier, which uses a raw value and not an enum.

Table 21: List of iPMF-3 Qualifiers

bcmFieldQualify…	Description
InPort	Ingress Port, value should be encoded as LOCAL GPORT Type. Includes core ID, and can be used by entries for all cores.
SrcPort	Qualify on source port, value should be encoded as SYS_PORT GPORT Type
DstPort	Qualify on destination port, all ingress stages expect raw value, egress expects GPORT encoded as SYS_PORT
InterfaceClassL2	VSI Profile
Vrf	VRF Id
SrcModPortGport	Qualify on Source module/port pair, value should be encoded as SYS_PORT GPORT Type
Color	Qualify based on packet color, value should be one of BCM_FIELD_COLOR_XXX
IntPriority	Internal priority (TC)
Vpn	VSI Id
FlowId	Qualify on destination Flow-Id
MplsForwardingLabelAction	MPLS forwarding label action
L2Learn	L2 learn enable
EcnValue	Qualify on metadata ECN/congestion information (including CNI)
RxTrapCode	Qualify on RX Trap Code
RxTrapData	Qualify on RX Trap qualifier
TrillEgressRbridge	Egress RBridge Nickname
ISid	I-SID (MAC-in-MAC lookup-id)
DstRpfGport	RPF destination (gport) for the RPF check
TrunkHashResult	Trunk Hash Result (i.e., the Load-balancing Key)
LearnSrcMac	Qualifies on learn source MAC
LearnVlan	Qualifies on learn VLAN
TranslatedOuterVlanId	Translated Outer VLAN ID
TranslatedOuterVlanPri	Translated Outer VLAN priority
TranslatedOuterVlanCfi	Translated Outer VLAN CFI
TranslatedInnerVlanId	Translated inner VLAN Id
TranslatedInnerVlanPri	Translated inner VLAN priority
TranslatedInnerVlanCfi	Translated inner VLAN CFI
DstGport	Qualifies packet according to destination encoded as trap code
RxTrapCodeForSnoop	Qualify on Snoop Code
IncomingIpIfClass	RIF profile
StackingRoute	Stacking Route History bitmap
RxTrapStrength	Qualify on RX Trap Strength

Table 21: List of iPMF-3 Qualifiers (Continued)

bcmFieldQualify…	Description
OamUpMep	It indicates if the OAM packet is vUP-MEP (sent to a destination in the network, as opposed to a specific port)
OamSubtype	In OAM the packet type is specified in the OAM header and mapped to a subtype in the hardware
OamHeaderOffset	indicates the offset of the OAM header relative to the start of packet (as opposed to start of header-offset)
OamStampOffset	indicates the offset to the position, in the OAM header, where the ToD or counter value should be stamped relative to the start of packet
OamMepId	If MEP is handled in OAMP, then the OAM-ID is the MEP-ID (equivalent to the index used to access the MEP DB)
VlanAction	VLAN translation action ID
GeneratedTtl	Qualify on Packet's generate Time-To-Live
PacketIsIEEE1588	Indicating whether the packet is 1588
IEEE1588Encapsulation	IEEE-1588 Encapsulation according to bcm_field_IEEE1588Encap_t
IEEE1588CompensateTimeStamp	IEEE-1588 update time stamp
IEEE1588Command	Command used by egress pipeline, indicating if CF(correction field) needs to be update
IEEE1588HeaderOffset	This field indicates the offset in bytes of the IEEE-1588 header
KeyGenVar	Match on configured key Gen Variable (values that was configured for context)
Container	This qualifier will be used as container in IPMF3, to receive the action buffer, when performing cascading between IPMF1/2 and IPMF3
UDHBase0	User-defined Header 0 encoded type and size – Value 1: type PMF size 32b
UDHBase1	User-defined Header 1 encoded type and size – Value 1: type PMF size 32b
UDHBase2	User-defined Header 2 encoded type and size – Value 1: type PMF size 32b
UDHBase3	User-defined Header 3 encoded type and size – Value 1: type PMF size 32b
NetworkQos	Qualify Upon Network QoS
ForwardingLayerIndex	Qualify on the Forwarding Layer Index
IPTProfile	Qualifies upon IPT Profile (End Of Packet Editing)
UDHData0	User-defined Header 0 payload MS bits of UDH
UDHData1	User-defined Header 1 payload
UDHData2	User-defined Header 2 payload
UDHData3	User-defined Header 3 payload LS bits of UDH
RxSnoopStrength	Snoop Strength
StatSamplingCode	Statistical Sampling Code value
StatSamplingQualifier	Statistical Sampling qualifier value
RpfEcmpMode	RPF ECMP mode
StatOamLM	OAM LM counter value – the index of the values are bcmFieldStatOamLmIndexXXX
InInterface	Ingress Logical Interface (LIF/RIF), value should be encoded as L3 interface
OutInterface	Egress Logical Interface (LIF/RIF), should be encoded as L3 interface
PortClassPacketProcessing	PP port profile
PortClassTrafficManagement	TM port profile
InVPort0	In LIF 0, value should be GPORT with subtype LIF
InVPort1	In LIF 1, value should be GPORT with subtype LIF
OutVPort0	OutLIF 0, value should be GPORT with subtype LIF
OutVPort1	OutLIF 1, value should be GPORT with subtype LIF
OutVPort2	OutLIF 2, value should be GPORT with subtype LIF

Table 21: List of iPMF-3 Qualifiers (Continued)

bcmFieldQualify…	Description
OutVPort3	OutLIF 3, value should be GPORT with subtype LIF
BierStringOffset	Qualifies upon Bier string offset
BierStringSize	Qualifies upon Bier string size
PacketIsBier	Qualifies upon Bier packets
PortClassPacketProcessing GeneralData	PP port general data
ForwardingAdditionalInfo	Qualifies on the Forwarding additional info
InVportClass0	InLIF Profile 0, by Gport
InVportClass1	InLIF Profile 1, by Gport
StatMetaData	Qualifies on Statistics metadata
InVPort0Raw	Qualifies on In LIF 0 input expects raw value
InVPort1Raw	Qualifies on In LIF 1 input expects raw value
OutVPort0Raw	Qualifies on OutLIF 0 input expects raw value
OutVPort1Raw	Qualifies on OutLIF 1 input expects raw value
OutVPort2Raw	Qualifies on OutLIF 2 input expects raw value
OutVPort3Raw	Qualifies on OutLIF 3 input expects raw value
MirrorCode	Matches on Mirror Code
MirrorData	Matches on Mirror Data (qualifier)
LearnVsi	Qualifies on learn VSI
LearnData	Qualifies on data field in the Learn Info
LearnStationMove	Qualifies on learn Station move
LearnMatch	Qualifies on learn Match
LearnFound	Qualifies on learn Found
LearnExpectedWon	Qualifies on learn Expected Won
VrfValue	Qualifies on the VRF part of forwarding data, without checking that the type is VRF
InPortWithoutCore	Ingress Port, value should be encoded as LOCAL GPORT Type. Does not include the core ID, caution should be taken when not adding entry by core
SrcPortRaw	Qualifies on source port, input expects raw value
InPortWithoutCoreRaw	Qualifies on Ingress Port. Does not include the core ID, caution should be taken when not adding entry by core, input expects raw value
InPortRaw	Qualifies on Ingress Port. Includes core ID, and can be used by entries for all cores, input expects raw value
ColorRaw	Qualifies on packet color, input expects raw value
DstRpfValid	Checks if the RPF Destination is valid
EcmpGroup	Qualifies on the ECMP group
StatOamLMRaw	OAM LM counter raw value

Table 22: List of Egress PMF Qualifiers

bcmFieldQualify…	Description
L4PortRangeCheck	Qualify on L4 Ops (L4 source and dest ports) range. The qualifier is a bitmap and not specific range id, it means if range 2 was configured and the qualifier is hit, the entry value is expected to be 0x100
SrcPort	Qualify on source port, value should be encoded as SYS_PORT GPORT Type

Table 22: List of Egress PMF Qualifiers (Continued)

bcmFieldQualify…	Description
DstPort	Qualify on destination port, all ingress stages expect raw value, egress expects GPORT encoded as SYS_PORT
SrcClassField	PPH Value 2
DstClassField	PPH Value 1
InterfaceClassL2	VSI Profile
Vrf	VRF Id
OutPort	Egress port, value should be encoded as LOCAL GPORT Type. Does not include the core ID, caution should be taken when not adding entry by core
DstL3Egress	L3 Egress Interface, Raw value of SYS_PORT
DstMulticastGroup	Multicast Group id, encoded as GPORT
SrcModPortGport	Qualify on Source module/port pair, value should be encoded as SYS_PORT GPORT Type
Color	Qualify based on packet color, value should be one of BCM_FIELD_COLOR_XXX
IntPriority	Internal priority (TC)
Vpn	VSI Id
RepCopy	Qualify on Copy type: 0:Forwarded / 1:Snooped / 2:Mirrored / 3:Statistical Sampling, packet
EcnValue	Qualify on metadata ECN or congestion information, including the CNI
RxTrapCode	Qualify on RX Trap Code
TrunkHashResult	Trunk Hash Result (i.e., the Load-balancing Key)
LearnSrcMac	Qualifies on learn source MAC
LearnVlan	Qualifies on learn VLAN
Fhei	DNX FHEI header field
FheiSize	DNX FHEI header size in bytes
StackingRoute	Stacking Route History bitmap
RxTrapStrength	Qualify on RX Trap Strength
OamTsSystemHeader	FTMH Application specific extension
DstSysPortExt	Qualifies upon TM Destination extension header
GeneratedTtl	Qualify on Packet's generate Time-To-Live
PphPresent	Qualifies on PPH present field in ITMH
PacketProcessingInVportClass	Qualifies on InLIF profile
UDHBase0	User-defined Header 0 encoded type and size – Value 1: type PMF size 32b
UDHBase1	User-defined Header 1 encoded type and size – Value 1: type PMF size 32b
UDHBase2	User-defined Header 2 encoded type and size – Value 1: type PMF size 32b
UDHBase3	User-defined Header 3 encoded type and size – Value 1: type PMF size 32b
NetworkQos	Qualify Upon Network QoS
AceEntryId	Qualify Upon Ace Entry ID
NetworkLoadBalanceKey	Network Load Balance Key
IPTProfile	Qualifies upon IPT Profile (End Of Packet Editing)
UDHData0	User-defined Header 0 payload MS bits of UDH
UDHData1	User-defined Header 1 payload
UDHData2	User-defined Header 2 payload
UDHData3	User-defined Header 3 payload LS bits of UDH
RxSnoopStrength	Snoop Strength
RxSnoopCode	qualify upon Snoop profile

Table 22: List of Egress PMF Qualifiers (Continued)

bcmFieldQualify…	Description
InInterface	Ingress Logical Interface (LIF/RIF), value should be encoded as L3 interface
OutInterface	Egress Logical Interface (LIF/RIF), should be encoded as L3 interface
OutVportClass	OutLIF Profile, by Gport
PortClassPacketProcessing	PP port profile
PortClassTrafficManagement	TM port profile
OutPortTrafficManagement	Out TM-port
InVPort0	In LIF 0, value should be GPORT with subtype LIF
OutVPort0	OutLIF 0, value should be GPORT with subtype LIF
OutVPort1	OutLIF 1, value should be GPORT with subtype LIF
OutVPort2	OutLIF 2, value should be GPORT with subtype LIF
OutVPort3	OutLIF 3, value should be GPORT with subtype LIF
ForwardingAdditionalInfo	Qualifies on the Forwarding additional info
InVportClass0	InLIF Profile 0, by Gport
InVPort0Raw	Qualifies on In LIF 0 input expects raw value
OutVPort0Raw	Qualifies on OutLIF 0 input expects raw value
OutVPort1Raw	Qualifies on OutLIF 1 input expects raw value
OutVPort2Raw	Qualifies on OutLIF 2 input expects raw value
OutVPort3Raw	Qualifies on OutLIF 3 input expects raw value
LearnVsi	Qualifies on learn VSI
LearnData	Qualifies on data field in the Learn Info
VrfValue	Qualifies on the VRF part of forwarding data, without checking that the type is VRF
DstSysPortExtPresent	Qualifies upon if TM Destination extension header is present
SrcPortRaw	Qualifies on source port, input expects raw value
ColorRaw	Qualifies on packet color, input expects raw value
DstPortRaw	Qualifies on destination port, input expects raw value
OutPortRaw	Qualifies on Egress port. Does not include the core ID, caution should be taken when not adding entry by core, input expects raw value
McastPkt	All ingress replicas will have this value set to 1
LearnExtensionPresent	Qualifies on whether the PPH learn extension is present
FtmhAsePresent	Qualifies the packet on the FTMH ASE present field

11.31.3 Actions List

To see latest version support, run diag: `field action Predefined stage=iPMF1/2/3/ePMF`.

Table 23: iPMF-1,2 Actions

bcmFieldAction…	Description
PrioIntNew	TC update
EcnNew	Change metadata ECN/congestion information (including CNI)
Redirect	Redirect packet to a destination
RedirectMcast	Redirect to Multicast Group Id
MirrorIngress	set mirror, value should be encoded as GPORT Type Mirror for mirror command
L3Switch	Sets packet destination, input expects L3 interface

Table 23: iPMF-1,2 Actions (Continued)

bcmFieldAction…	Description
DropPrecedence	Set Packet's Drop Precedence, the values are BCM_FIELD_COLOR_XXX
VrfSet	Set VRF. value[0]: fwd_domain_id_type. value[1]: fwd_domain_id_value. value[1]: fwd_domain_profile. In iPMF3, fwd_domain_profile isn't updated
InnerVlanNew	Replace inner VLAN Id
OuterVlanNew	Replace outer VLAN Id
OuterVlanPrioNew	Replace outer VLAN tag priority
VSQ	Assign matching packets to specified VSQ
Trap	Set Trap code, use TRAP GPORT type value[0]=TRAP_GPORT value[1]=Trap Qualifier. at egress stage refers only to the TRAP ID (strength is not relevant, and can be changed using bcmFieldActionTrapStrength action)
UsePolicerResult	Specify/override where policer result will be used for matched packets
Snoop	Snoop matched packets, input can be either a snoop command: value[0] = Mirror Gport, value[1]=snoop qualifier, value[2]=snoop strength OR: snoop code: value[0] = Trap Gport, value[1]=snoop qualifier, at egress stage refers only to the snoop ID (strength is not relevant, and can be changed using bcmFieldActionSnoopStrength action)
TtlSet	New TTL
DstRpfGportNew	Set the RPF Destination
SrcGportNew	Set the Source-Port
SystemHeaderSet	Modify EEI
VSwitchNew	New VSI. value[0]: fwd_domain_id_type. value[1]: fwd_domain_id_value. value[1]: fwd_domain_profile. In iPMF3, fwd_domain_profile isn't updated
VlanActionSetNew	Modify the VLAN Action Set Id
IngressGportSet	Set a new ingress Global Lif encoded, Values should be Encoded as GPORT
StackingRouteNew	Replace the value of the stacking route
PphPresentSet	If set, a packet processing header is present
FabricHeaderSet	Change System Header Profile, bcm_field_system_header_profile_t
Oam	Changes 4 OAM signals. value[0]: OAM-Stamp-Offset. value[1]: OAM-offset. value[2]: OAM-Sub-Type. value[3]: OAM-Up-Mep
PacketTraceEnable	Enable Trace packet
IEEE1588	Setting various parameters for 1588 frames. Bits(0:7): header_offset. Bit(8): encapsulation. Bits(9:10) command. Bit(11): compensate_time_stamp. Bit(12): packet_is_ieee1588
Forward	Forward Destination Raw value, setting to 1's will drop the packet
PphSnoopCode	Set the PPH Snoop code value (Reserved Bits)
IngressTimeStampInsert	Insert IPIPE time stamp
AdmitProfile	Admit Profile
LatencyFlowId	Change the latency flow-Id. value[0] valid. value[1]: latency_flow_id. value[2]: latency_flow_profile
VisibilityEnable	Enables visibility for current packet
InVportClass0	Updates the profile of an in LIF encoded as Gport
InVportClass1	Updates the profile of an in LIF encoded as Gport
NetworkQos	Updates Network Qos value
Void	Void action, used for cascading, saving buffer in the action payload (for FEM use)
Container	This action will be used as container in IPMF2, to parse the action buffer, when performing cascading between IPMF2 and IPMF3
EgressForwardingIndex	Indicate for egress parser which layer is the Forwarding layer
UDHData0	Set the Payload for User-defined Header 0

Table 23: iPMF-1,2 Actions (Continued)

bcmFieldAction…	Description
UDHData1	Set the Payload for User-defined Header 1
UDHData2	Set the Payload for User-defined Header 2
UDHData3	Set the Payload for User-defined Header 3
UDHBase0	Set the Base for User-defined Header 0
UDHBase1	Set the Base for User-defined Header 1
UDHBase2	Set the Base for User-defined Header 2
UDHBase3	Set the Base for User-defined Header 3
IPTProfile	IPT Profile action (used for instrumentation)
StatId0	Updates statistics ID 0
StatId1	Updates statistics ID 1
StatId2	Updates statistics ID 2
StatId3	Updates statistics ID 3
StatId4	Updates statistics ID 4
StatId5	Updates statistics ID 5
StatId6	Updates statistics ID 6
StatId7	Updates statistics ID 7
StatId8	Updates statistics ID 8
StatId9	Updates statistics ID 9
StatProfile0	Updates statistics profile 0. value[0]: is_meter. value[1]: is_lm. value[2]: type. value[3]: valid
StatProfile1	Updates statistics profile 1. value[0]: is_meter. value[1]: is_lm. value[2]: type. value[3]: valid
StatProfile2	Updates statistics profile 2. value[0]: is_meter. value[1]: is_lm. value[2]: type. value[3]: valid
StatProfile3	Updates statistics profile 3. value[0]: is_meter. value[1]: is_lm. value[2]: type. value[3]: valid
StatProfile4	Updates statistics profile 4. value[0]: is_meter. value[1]: is_lm. value[2]: type. value[3]: valid
StatProfile5	Updates statistics profile 5. value[0]: is_meter. value[1]: is_lm. value[2]: type. value[3]: valid
StatProfile6	Updates statistics profile 6. value[0]: is_meter. value[1]: is_lm. value[2]: type. value[3]: valid
StatProfile7	Updates statistics profile 7. value[0]: is_meter. value[1]: is_lm. value[2]: type. value[3]: valid
StatProfile8	Updates statistics profile 8. value[0]: is_meter. value[1]: is_lm. value[2]: type. value[3]: valid
StatProfile9	Updates statistics profile 9. value[0]: is_meter. value[1]: is_lm. value[2]: type. value[3]: valid
InvalidNext	Invalid next Action macro Id
IngressDoNotLearn	Disable Ingress learning
EgressDoNotLearn	Disable Egress learning
VisibilityClear	Disables visibility for current packet
StatMetaData	Set statistics metaData
InVport0	Change the InLIF 0, value should be encoded as a GPORT with LIF subtype
InVport1	Change the InLIF 1, value should be encoded as a GPORT with LIF subtype
OutVport0	Change OutLIF 0, value should be encoded as a GPORT with LIF subtype
OutVport1	Change OutLIF 1, value should be encoded as a GPORT with LIF subtype
OutVport2	Change OutLIF 2, value should be encoded as a GPORT with LIF subtype
OutVport3	Change OutLIF 3, value should be encoded as a GPORT with LIF subtype
ForwardingTypeMeterMap	Maps L2 forwarding type to one of the general meters
StatSampling	Update statistical sampling value, value encoded as Mirror Gport
StatOamLM	Update the counter of OAM LM, the index of the values are bcmFieldStatOamLmIndexXXX
ForwardingLayerIndex	Update forwarding layer index

Table 23: iPMF-1,2 Actions (Continued)

bcmFieldAction…	Description
InInterface0	Change the InLIF 0, value should be encoded as an interface
InInterface1	Change the InLIF 1, value should be encoded as an interface
OutInterface0	Change OutLIF 0, value should be encoded as type interface
OutInterface1	Change OutLIF 1, value should be encoded as type interface
OutInterface2	Change OutLIF 2, value should be encoded as type interface
OutInterface3	Change OutLIF 3, value should be encoded as type interface
BierStringOffset	Updates Bier string offset
BierStringSize	Updates Bier string size
PacketIsBier	Updates is Bier packet
Eventor	Enable push data to eventor
ForwardingAdditionalInfo	Updates forwarding additional information
SmallExemLearn	Updates the Small Exact Match learn info (IPMF2 only)
InVport0Raw	Change the InLIF 0, with raw value
InVport1Raw	Change the InLIF 1, with raw value
OutVport0Raw	Change the OutLIF 0, with raw value
OutVport1Raw	Change the OutLIF 1, with raw value
OutVport2Raw	Change the OutLIF 2, with raw value
OutVport3Raw	Change the OutLIF 3, with raw value
IngressTimeStampInsertValid	Insert ingress timestamp. Overrides the time stamp in TSH header
SnoopRaw	Snoop matched packets, input should be raw value. At egress stage refers only to the snoop ID (strength is not relevant, and can be changed using bcmFieldActionSnoopStrength action)
DropPrecedenceRaw	Set Packet's Drop Precedence, input expects raw value
FabricHeaderSetRaw	Change System Header Profile, input expects raw value
SrcGportNewRaw	Set the Source-Port, input expects raw value
LatencyFlowIdRaw	Change the latency flow-Id, input expects raw value
MirrorIngressRaw	Set mirror in ingress, input expects raw value
OamRaw	Changes 4 OAM signals (OAM-Stamp-Offset, OAM-offset, OAM-Sub-Type, OAM-Up-Mep), input expects raw value
TrapRaw	Set Trap code, input should be raw value. At egress stage refers only to the TRAP ID (strength is not relevant, and can be changed using bcmFieldActionTrapStrength action)
VrfSetRaw	Set VRF, input expects raw value
StatProfile0Raw	Updates statistics profile 0. Input expects raw value
StatProfile1Raw	Updates statistics profile 1. Input expects raw value
StatProfile2Raw	Updates statistics profile 2. Input expects raw value
StatProfile3Raw	Updates statistics profile 3. Input expects raw value
StatProfile4Raw	Updates statistics profile 4. Input expects raw value
StatProfile5Raw	Updates statistics profile 5. Input expects raw value
StatProfile6Raw	Updates statistics profile 6. Input expects raw value
StatProfile7Raw	Updates statistics profile 7. Input expects raw value
StatProfile8Raw	Updates statistics profile 8. Input expects raw value
StatProfile9Raw	Updates statistics profile 9. Input expects raw value
StatSamplingRaw	Update statistical sampling value, input expects raw value
IpMulticastCompatible	Designates when a Compatible MC procedure found a match

Table 23: iPMF-1,2 Actions (Continued)

bcmFieldAction…	Description
RpfOutVport	Updates the RPF OUT LIF. value[0]: route_valid, value[1]: default_route_found, value[2]: rpf_out_lif (encoded as GPORT with LIF subtype)
RpfOutInterface	Updates the RPF OUT LIF. value[0]: route_valid, value[1]: default_route_found, value[2]: rpf_out_lif (encoded as L3 interface)
RpfOutVportRaw	Updates the RPF OUT LIF. Input expects raw value
LearnRaw0	Updates learn info 0
LearnRaw1	Updates learn info 1
LearnRaw2	Updates learn info 2
LearnRaw3	Updates learn info 3
LearnRaw4	Updates learn info 4
DstRpfGportNewValid	Set the RPF Destination Valid
DstRpfGportNewRaw	Set the RPF Destination. Input expects raw value
StatOamLMRaw	Updates the counter of OAM LM with a raw value
LearnKey0	Updates bits 31:0 of learn info key
LearnKey1	Updates bits 63:32 of learn info key
LearnKey2	Updates learn info key and app_db_index. value[0]: bits 79:64 of learn info ley, value[1] app_db_index
LearnKey2Raw	Updates learn info key and app_db_index. Key is at the LSB
LearnPayload0	Updates bits 31:0 of learn info data (payload)
LearnPayload1	Updates bits 63:32 of learn info data (payload)
LearnEntryFormatIndex	Updates entry format index in learn info
LearnOrTransplant	Updates learn or transplant bit

Table 24: iPMF-3 Actions

bcmFieldAction…	Description
PrioIntNew	TC update
EcnNew	Change metadata ECN or congestion information (including CNI)
Redirect	Redirect packet to destination
RedirectMcast	Redirect to Multicast Group Id
MirrorIngress	set mirror, value should be encoded as GPORT Type Mirror for mirror command
L3Switch	Sets packet destination, input expects L3 interface
DropPrecedence	Set Packet's Drop Precedence, the values are BCM_FIELD_COLOR_XXX
VrfSet	Set VRF. value[0]: fwd_domain_id_type. value[1]: fwd_domain_id_value. value[1]: fwd_domain_profile. In iPMF3, fwd_domain_profile isn't updated
InnerVlanNew	Replace inner VLAN Id
OuterVlanNew	Replace outer VLAN Id
OuterVlanPrioNew	Replace outer VLAN tag priority
VSQ	Assign matching packets to specified VSQ
Trap	Set Trap code, use TRAP GPORT type value[0]=TRAP_GPORT value[1]=Trap Qualifier. at egress stage refers only to the TRAP ID (strength is not relevant, and can be changed using bcmFieldActionTrapStrength action)
UsePolicerResult	Specify/override where policer result will be used for matched packets

Table 24: iPMF-3 Actions (Continued)

bcmFieldAction…	Description
Snoop	Snoop matched packets, input can be either a snoop command: value[0] = Mirror Gport, value[1]=snoop qualifier, value[2]=snoop strength OR: snoop code: value[0] = Trap Gport, value[1]=snoop qualifier, at egress stage refers only to the snoop ID (strength is not relevant, and can be changed using bcmFieldActionSnoopStrength action)
TtlSet	New TTL
TrunkHashKeySet	Set the Trunk Hash Key
SrcGportNew	Set the Source-Port
StartPacketStrip	Strip the start of packet. value[0]: bcm_field_start_packet_strip_t value[1]: extra bytes to remove
SystemHeaderSet	Modify EEI
VSwitchNew	New VSI. value[0]: fwd_domain_id_type. value[1]: fwd_domain_id_value. value[1]: fwd_domain_profile. In iPMF3, fwd_domain_profile isn't updated
VlanActionSetNew	Modify the VLAN Action Set Id
IngressGportSet	Set a new ingress Global Lif encoded, Values should be Encoded as GPORT
StackingRouteNew	Replace the value of the stacking route
PphPresentSet	If set, a packet processing header is present
FabricHeaderSet	Change System Header Profile, bcm_field_system_header_profile_t
Oam	Changes 4 OAM signals. value[0]: OAM-Stamp-Offset. value[1]: OAM-offset. value[2]: OAM-Sub-Type. value[3]: OAM-Up-Mep
IEEE1588	Setting various parameters for 1588 frames. Bits(0:7): header_offset. Bit(8): encapsulation. Bits(9:10) command. Bit(11): compensate_time_stamp. Bit(12): packet_is_ieee1588
Forward	Forward Destination Raw value, setting to 1's will drop the packet
PphSnoopCode	Set the PPH Snoop code value (Reserved Bits)
AdmitProfile	Admit Profile
LatencyFlowId	Change the latency flow-Id. value[0] valid. value[1]: latency_flow_id. value[2]: latency_flow_profile
VisibilityEnable	Enables visibility for current packet
SystemHeaderSizeAdjust	System Header Size adjust
InVportClass0	Updates the profile of an in LIF encoded as Gport
InVportClass1	Updates the profile of an in LIF encoded as Gport
NetworkQos	Updates Network Qos value
Void	Void action, used for cascading, saving buffer in the action payload (for FEM use)
ParsingStartType	Indicates the layer type that egress parser should consider. For information about the possible layer types, see Section 11.30, Common Types
ParsingStartOffset	Indicate from which offset egress parser should start parsing (all info before that will not be parsed)
EgressForwardingIndex	Indicate for egress parser which layer is the Forwarding layer
UDHData0	Set the Payload for User-defined Header 0
UDHData1	Set the Payload for User-defined Header 1
UDHData2	Set the Payload for User-defined Header 2
UDHData3	Set the Payload for User-defined Header 3
UDHBase	Set the Base for All User-defined Header 0-3, UDHBase0 is at LSB
IPTProfile	IPT Profile action (used for instrumentation)
IPTCommand	IPT Profile command. value[0]: int_commnad. value[1]: int_profile
StatId0	Updates statistics ID 0
StatId1	Updates statistics ID 1
StatId2	Updates statistics ID 2

Table 24: iPMF-3 Actions (Continued)

bcmFieldAction…	Description
StatId3	Updates statistics ID 3
StatId4	Updates statistics ID 4
StatId5	Updates statistics ID 5
StatId6	Updates statistics ID 6
StatId7	Updates statistics ID 7
StatId8	Updates statistics ID 8
StatId9	Updates statistics ID 9
StatProfile0	Updates statistics profile 0. value[0]: is_meter. value[1]: is_lm. value[2]: type. value[3]: valid
StatProfile1	Updates statistics profile 1. value[0]: is_meter. value[1]: is_lm. value[2]: type. value[3]: valid
StatProfile2	Updates statistics profile 2. value[0]: is_meter. value[1]: is_lm. value[2]: type. value[3]: valid
StatProfile3	Updates statistics profile 3. value[0]: is_meter. value[1]: is_lm. value[2]: type. value[3]: valid
StatProfile4	Updates statistics profile 4. value[0]: is_meter. value[1]: is_lm. value[2]: type. value[3]: valid
StatProfile5	Updates statistics profile 5. value[0]: is_meter. value[1]: is_lm. value[2]: type. value[3]: valid
StatProfile6	Updates statistics profile 6. value[0]: is_meter. value[1]: is_lm. value[2]: type. value[3]: valid
StatProfile7	Updates statistics profile 7. value[0]: is_meter. value[1]: is_lm. value[2]: type. value[3]: valid
StatProfile8	Updates statistics profile 8. value[0]: is_meter. value[1]: is_lm. value[2]: type. value[3]: valid
StatProfile9	Updates statistics profile 9. value[0]: is_meter. value[1]: is_lm. value[2]: type. value[3]: valid
InvalidNext	Invalid next Action macro Id
StatMetaData	Set statistics metaData
InVport0	Change the InLIF 0, value should be encoded as a GPORT with LIF subtype
InVport1	Change the InLIF 1, value should be encoded as a GPORT with LIF subtype
OutVport0	Change OutLIF 0, value should be encoded as a GPORT with LIF subtype
OutVport1	Change OutLIF 1, value should be encoded as a GPORT with LIF subtype
OutVport2	Change OutLIF 2, value should be encoded as a GPORT with LIF subtype
OutVport3	Change OutLIF 3, value should be encoded as a GPORT with LIF subtype
ForwardingTypeMeterMap	Maps L2 forwarding type to one of the general meters
StatSampling	Update statistical sampling value, value encoded as Mirror Gport
StatOamLM	Update the counter of OAM LM, the index of the values are bcmFieldStatOamLmIndexXXX
ForwardingLayerIndex	Update forwarding layer index
InInterface0	Change the InLIF 0, value should be encoded as an interface
InInterface1	Change the InLIF 1, value should be encoded as an interface
OutInterface0	Change OutLIF 0, value should be encoded as type interface
OutInterface1	Change OutLIF 1, value should be encoded as type interface
OutInterface2	Change OutLIF 2, value should be encoded as type interface
OutInterface3	Change OutLIF 3, value should be encoded as type interface
BierStringOffset	Updates Bier string offset
BierStringSize	Updates Bier string size
PacketIsBier	Updates is Bier packet
ForwardingAdditionalInfo	Updates forwarding additional information
InVport0Raw	Change the InLIF 0, with raw value
InVport1Raw	Change the InLIF 1, with raw value
OutVport0Raw	Change the OutLIF 0, with raw value
OutVport1Raw	Change the OutLIF 1, with raw value

Table 24: iPMF-3 Actions (Continued)

bcmFieldAction…	Description
OutVport2Raw	Change the OutLIF 2, with raw value
OutVport3Raw	Change the OutLIF 3, with raw value
AppendPointerCompensation	Configures pointer to header append compensation value
IngressTimeStampInsertValid	Insert ingress timestamp. Overrides the time stamp in TSH header
SnoopRaw	Snoop matched packets, input should be raw value. At egress stage refers only to the snoop ID (strength is not relevant, and can be changed using bcmFieldActionSnoopStrength action)
DropPrecedenceRaw	Set Packet's Drop Precedence, input expects raw value
FabricHeaderSetRaw	Change system header profile, input expects raw value
SrcGportNewRaw	Set the Source-Port, input expects raw value
LatencyFlowIdRaw	Change the latency flow-Id, input expects raw value
MirrorIngressRaw	Set mirror in ingress, input expects raw value
OamRaw	Changes four OAM signals (OAM-Stamp-Offset, OAM-offset, OAM-Sub-Type, OAM-Up-Mep), input expects raw value
TrapRaw	Set Trap code, input should be raw value. At egress stage refers only to the TRAP ID (strength is not relevant, and can be changed using bcmFieldActionTrapStrength action)
StatProfile0Raw	Updates statistics profile 0. Input expects raw value
StatProfile1Raw	Updates statistics profile 1. Input expects raw value
StatProfile2Raw	Updates statistics profile 2. Input expects raw value
StatProfile3Raw	Updates statistics profile 3. Input expects raw value
StatProfile4Raw	Updates statistics profile 4. Input expects raw value
StatProfile5Raw	Updates statistics profile 5. Input expects raw value
StatProfile6Raw	Updates statistics profile 6. Input expects raw value
StatProfile7Raw	Updates statistics profile 7. Input expects raw value
StatProfile8Raw	Updates statistics profile 8. Input expects raw value
StatProfile9Raw	Updates statistics profile 9. Input expects raw value
StatSamplingRaw	Update statistical sampling value, input expects raw value
StartPacketStripRaw	Strip the start of packet. Refers to bcm_field_start_packet_strip_t. Input expects raw value
ParsingStartTypeRaw	Indicate the layer type that egress parser should consider, input expects raw value.
ParsingStartOffsetRaw	Indicate from which offset egress parser should start parsing (all info before that will not be parsed). Input expects raw value
IPTCommandRaw	IPT Profile command. Input expects raw value
LearnRaw0	Updates learn info 0
LearnRaw1	Updates learn info 1
LearnRaw2	Updates learn info 2
LearnRaw3	Updates learn info 3
LearnRaw4	Updates learn info 4
StatOamLMRaw	Update the counter of OAM LM with a raw value
LearnKey0	Updates bits 31:0 of learn info key
LearnKey1	Updates bits 63:32 of learn info key
LearnKey2	Updates learn info key and app_db_index. value[0]: bits 79:64 of learn info ley, value[1] app_db_index
LearnKey2Raw	Updates learn info key and app_db_index. Key is at the LSB
LearnPayload0	Updates bits 31:0 of learn info data (payload)
LearnPayload1	Updates bits 63:32 of learn info data (payload)
LearnEntryFormatIndex	Updates entry format index in learn info

Table 24: iPMF-3 Actions (Continued)

bcmFieldAction…	Description
LearnOrTransplant	Updates learn or transplant bit
NetworkLoadBalanceKey	Set the network load balancing Key

Table 25: Egress-PMF Actions

bcmFieldAction…	Description
PrioIntNew	TC update **NOTE:** In the ePMF, setting MC traffic to TC=4 is not allowed and can cause traffic issues. Do not use this setting in that case.
Drop	Discard packet
MirrorEgress	Update the mirror profile, mirror_cmd should be encoded as MIRROR Gport
DropPrecedence	Set Packet's Drop Precedence, the values are BCM_FIELD_COLOR_XXX
Trap	Set Trap code, use TRAP GPORT type value[0]=TRAP_GPORT value[1]=Trap Qualifier. at egress stage refers only to the TRAP ID (strength is not relevant, and can be changed using bcmFieldActionTrapStrength action)
Snoop	Snoop matched packets, input can be either a snoop command: value[0] = Mirror Gport, value[1]=snoop qualifier, value[2]=snoop strength OR: snoop code: value[0] = Trap Gport, value[1]=snoop qualifier, at egress stage refers only to the snoop ID (strength is not relevant, and can be changed using bcmFieldActionSnoopStrength action)
QosMapIdNew	Set the QoS profile
Void	Void action, used for cascading, saving buffer in the action payload (for FEM use)
StatId0	Updates statistics ID 0
StatId1	Updates statistics ID 1
StatProfile0	Updates statistics profile 0. value[0]: is_meter. value[1]: is_lm. value[2]: type. value[3]: valid
StatProfile1	Updates statistics profile 1. value[0]: is_meter. value[1]: is_lm. value[2]: type. value[3]: valid
InvalidNext	Invalid next Action macro Id
AceEntryId	Pointer to the ACE entry ID
TrapStrength	Updates the Trap Strength. Use TRAP Gport
SnoopStrength	Updates the snoop strength
OutPortTrafficManagement	Updates the Out-TM port
SnoopRaw	Snoop matched packets, input should be raw value. At egress stage refers only to the snoop ID (strength is not relevant, and can be changed using bcmFieldActionSnoopStrength action)
DropPrecedenceRaw	Set Packet's Drop Precedence, input expects raw value
MirrorEgressRaw	Update the egress mirror profile, input expects raw value
TrapStrengthRaw	Updates the Trap Strength, input expects raw value
SnoopStrengthRaw	Updates the Snoop Strength, input expects raw value

Table 26: Egress Extension (ACE) PMF Actions

bcmFieldAction…	Description
PrioIntNew	TC update
EcnNew	Change metadata ECN or congestion information (including CNI)
DropPrecedence	Set Packet's Drop Precedence, the values are BCM_FIELD_COLOR_XXX
Stat0	Stat meter object. value[0]: stat_meter_obj_cmd. value[1]: stat_meter_obj_id
Stat1	Stat counter object. value[0]: stat_counter_obj_cmd. value[1]: stat_counter_obj_id. value[2]: valid

Table 26: Egress Extension (ACE) PMF Actions (Continued)

bcmFieldAction…	Description
TtlSet	New TTL
LearnSrcPortNew	Replace the learned packet source Port
InterfaceClassVPort	Set InLIF profile
Void	Void action, used for cascading, saving buffer in the action payload (for FEM use)
IPTProfile	IPT Profile action (used for instrumentation)
InvalidNext	Invalid next Action macro Id
OutVport0	Change OutLIF 0, value should be encoded as a GPORT with LIF subtype
OutVport1	Change OutLIF 1, value should be encoded as a GPORT with LIF subtype
OutVport2	Change OutLIF 2, value should be encoded as a GPORT with LIF subtype
OutVport3	Change OutLIF 3, value should be encoded as a GPORT with LIF subtype
ForwardingLayerIndex	Update forwarding layer index
OutInterface0	Change OutLIF 0, value should be encoded as type interface
OutInterface1	Change OutLIF 1, value should be encoded as type interface
OutInterface2	Change OutLIF 2, value should be encoded as type interface
OutInterface3	Change OutLIF 3, value should be encoded as type interface
AceContextValue	Updates programmable value per ACE context, bcm_field_ace_context_t. Up to four different values can be given in value[0-3], although some may be mutually exclusive
ForwardingAdditionalInfo	Updates forwarding additional information
OutVport0Raw	Change the OutLIF 0, with raw value
OutVport1Raw	Change the OutLIF 1, with raw value
OutVport2Raw	Change the OutLIF 2, with raw value
OutVport3Raw	Change the OutLIF 3, with raw value
DropPrecedenceRaw	Set Packet's Drop Precedence, input expects raw value
AceContextValueRaw	Updates programmable value per ACE context, input expects raw value
Stat0Raw	Stat meter object. Input expects raw value
Stat1Raw	Stat counter object. Input expects raw value

NOTE: Action sizes can be obtained by using the following shell command:

```
field map action bcm <stage=ipmf1/2/3..epmf>.
```

11.32 Information Specific to the BCM88480 (Q2A) Device

The following table shows the number of various resources per stage.

Resource	Stage			
	iPMF-1	**iPMF-2**	**iPMF-3**	**ePMF**
Context selection lines	128	128	128	128
Contexts	64		64	64
Total number of keys	5 + 5 initial keys	5	3	4
TCAM lookup keys	4	4	2	2
Direct extraction keys	0	2	1	1
Exact match keys	1	1		1
FES contexts	32		16	12
FEM	16		0	0

Table 27: BCM88480 Database Properties

Type	TCAM Lookup
Maximum key size	Possible key sizes
	80b/160b/320b
Maximum payload size	32b/64b/128b
Memory	Shared (with other TCAM FGs)
Maximum number of entries per FG	25K/12.5K/6.25K
Supported stages	iPMF1/2/3 ePMF
Maximum FGs (in a given stage) per context	4/4/2 – iPMF1/2 2/2/1 – iPMF3 2/2/1 – ePMF **NOTE:** For ePMF, double lookup key size must be > 160b.

11.32.1 CS Qualifiers

For a list of context selection qualifiers, see Section 11.31.1, CS Qualifiers.

To see latest version support, run the following diagnostic command:

```
field qualifier ContextSelection
```

Table 28: List of iPMF1 Stage Context Selection Qualifiers

bcmFieldQualify…	Description
ForwardingType	Use this qualifier together with `input_arg` to indicate which layer number relative to forwarding layer to qualify upon. Possible layer types are bcmFieldLayerTypeXXX.
VlanFormat	Incoming vlan structure format (untagged/single/double/priority-tagged). For possible values, see bcm_port_tag_struct_type_t.

Table 28: List of iPMF1 Stage Context Selection Qualifiers (Continued)

bcmFieldQualify…	Description
RxTrapCode	Qualify on RX Trap Code
Ptch	Opaque attribute field of the Injected packets. Part of the PTCH_2 header. The input expects a raw value.
VPortRangeCheck	Virtual Port Range Check Results
AppType	Packet Application type used by the ACL lookup. Can be either predefined (bcm_field_AppType_t) or user defined
ForwardingLayerIndex	Qualify on the Forwarding Layer Index
ForwardingLayerQualifier	Use this qualifier together with `input_arg` to indicate which layer number relative to forwarding layer to qualify upon. It qualifies Layer Qualifiers (look for layer record qualifiers in UM)
TracePacket	Qualifies upon Packets that Trace was set for them
InVportClass	Qualify on the profile of an in LIF encoded as Gport
PortClassPacketProcessing	PP port profile
PortClassTrafficManagement	TM port profile
AppTypePredefined	The predefined AppType (bcm_field_AppType_t) used by the forwarding lookup
VlanFormatRaw	Qualifies on the incoming VLAN structure format (untagged, single tagged, double-tagged, or priority-tagged). The input expects a raw value.
AppTypeRaw	Qualifies on the packet application type used by the ACL lookup. Can be either predefined or user-defined. The input expects a raw value.
AppTypePredefinedRaw	Qualifies on the predefined AppType used by the forwarding lookup. The input expects a raw value.
ForwardingTypeRaw	The same qualifier as bcmFieldQualifyForwardingType but does not have a mapping function. In other words, the input expects a raw value (HW value).
PrtQualifier	Qualifies on the prt_qualifier signal, which may be set by the injected packet (Opaque attribute field part of the PTCH_2 header). The input expects an enum value `bcm_field_prt_qualifier_t`. Similar to the ptch qualifier, which uses a raw value and not an enum.
AcInLifWideData	Qualifies on the Ac InLIF wide (generic) data.

Table 29: List of iPMF2 Stage Context Selection Qualifiers

bcmFieldQualify…	Description
CascadedKeyValue	Value of key cascaded from prior group in cascade
CompareKeysResult0	compare keys result 0
CompareKeysResult1	compare keys result 1

Table 30: List of iPMF3 Stage Context Selection Qualifiers

bcmFieldQualify…	Description
ForwardingType	Qualify upon Forwarding Layer Type. Possible layer types are: bcmFieldLayerTypeXXX
VPortRangeCheck	Virtual Port Range Check Results
ContextId	Qualifies on the Context Id
ForwardingLayerQualifier	Use this qualifier together with `input_arg` to indicate which layer number relative to forwarding layer to qualify upon. It qualifies Layer Qualifiers (look for layer record qualifiers in UM).
RpfEcmpMode	RPF ECMP mode
PortClassPacketProcessing	PP port profile
PortClassTrafficManagement	TM port profile

Table 30: List of iPMF3 Stage Context Selection Qualifiers (Continued)

bcmFieldQualify…	Description
ForwardingTypeRaw	The same qualifier as bcmFieldQualifyForwardingType but does not have a mapping function. In other words, the input expects a raw value (HW value).

Table 31: List of Egress-PMF Stage Context Selection Qualifiers

bcmFieldQualify…	Description
SrcClassField	PPH Value 2
DstClassField	PPH Value 1
ForwardingType	Use this qualifier together with `input_arg` to indicate which layer number relative to forwarding layer to qualify upon. Possible layer types are bcmFieldLayerTypeXXX.
VlanFormat	Incoming vlan structure format (untagged/single/double/priority-tagged). For possible values, see bcm_port_tag_struct_type_t.
FheiSize	DNX FHEI header size in bytes
PphPresent	Qualifies on PPH present field in ITMH
UDHBase0	User-defined Header 0 encoded type and size – Value 1: type PMF size 32b
UDHBase1	User-defined Header 1 encoded type and size – Value 1: type PMF size 32b
UDHBase2	User-defined Header 2 encoded type and size – Value 1: type PMF size 32b
UDHBase3	User-defined Header 3 encoded type and size – Value 1: type PMF size 32b
ContextId	Qualifies on the Context Id (see bcmFIeldForwardContextXXX)
ForwardingLayerQualifier	Use this qualifier together with `input_arg` to indicate which layer number relative to forwarding layer to qualify upon. It qualifies Layer Qualifiers (look for layer record qualifiers in UM).
PortClassPacketProcessing	PP port profile
InVportClass0	InLIF Profile 0, by gport
VlanFormatRaw	Qualifies on the incoming VLAN structure format (untagged, single-tagged, double-tagged, or priority-tagged). The input expects a raw value.
LearnExtensionPresent	Qualifies on whether a PPH learn extension is present.
ForwardingTypeRaw	The same qualifier as bcmFieldQualifyForwardingType but without the mapping function. That is, the input expects a raw value (HW value).
ContextIdRaw	Qualifies on the forwarding context value. The input expects a raw value. Relevant for egress stage only.
FtmhAsePresent	Qualify packet on FTMH ASE present field.
FtmhAseType	Qualify packet on FTMH ASE type value.

11.32.2 Qualifiers List

To see latest version support, run the following diagnostic command:

```
field qualifier Predefined bcm stage=iPMF1/2/3/ePMF class=meta/layer/header
```

11.32.2.1 Header Qualifiers List

Table 32: Header Qualifiers List

bcmFieldQualify…	Description
SrcIp6	Source IPv6 address
DstIp6	Destination IPv6 address
SrcIp6High	Source IPv6 address (High/Upper 64 bits, 64 MSB)

Table 32: Header Qualifiers List (Continued)

bcmFieldQualify…	Description
DstIp6High	Destination IPv6 address (High/Upper 64 bits, 64 MSB)
SrcIp6Low	Source IPv6 address (Low/Lower 64 bits, 64 LSB)
DstIp6Low	Destination IPv6 address (Low/Lower 64 bits, 64 LSB)
SrcMac	Source MAC address
DstMac	Destination MAC address
SrcIp	Packet source IP address
DstIp	Packet destination IP address
Ip6FlowLabel	IPv6 flow label
L4SrcPort	L4 source port
L4DstPort	L4 destination port
Ip6NextHeader	Next protocol field in IPv6 header
Ip6TrafficClass	Traffic class field in IPv6 header
Ip6HopLimit	Hop count field in IPv6 header
TcpControl	TCP control flags
IpFlags	IP flags field
ExtensionHeaderType	Qualifies on Next Header Field in First Extension Header
ArpSenderIp4	Sender IPv4 field of ARP header
ArpTargetIp4	Target IPv4 field of ARP header
ArpOpcode	Opcode field of ARP header
Ip4Protocol	Qualify upon IPv4 protocol
Ip4Tos	Qualify upon IPv4 TOS
Ip4Ttl	Qualify upon IPv4 TTL
EtherTypeUntagged	802.1 Ethernet Type; only for untagged frames
Tpid	VLAN TPID, offset of the value is inside the VLAN tag. Add the offset where the tag start in the layer (inner/outer).
VlanId	VLAN ID, offset of the value is inside the VLAN tag. Add the offset where the tag start in the layer (inner/outer).
VlanPri	VLAN P-bits, offset of the value is inside the VLAN tag. Add the offset where the tag start in the layer (inner/outer).
VlanCfi	VLAN CFI, offset of the value is inside the VLAN tag. Add the offset where the tag start in the layer (inner/outer).
VlanPriCfi	VLAN P-bits and CFI, offset of the value is inside the VLAN tag. Add the offset where the tag start in the layer (inner/outer).
MplsLabel	MPLS label
MplsLabelId	ID field of MPLS label
MplsLabelTtl	TTL field of MPLS label
MplsLabelBos	BOS field of MPLS label
MplsLabelExp	TTL field of MPLS label

11.32.2.2 Layer Record Qualifiers List

Table 33: Layer Record Qualifiers List

bcmFieldQualify…	Description
IpFrag	IP fragments
LayerRecordType	Qualifies on the Layer Record Type. For possible values look at bcm_field_layer_type_t.

Table 33: Layer Record Qualifiers List (Continued)

bcmFieldQualify…	Description
LayerRecordOffset	Qualifies on the Layer Record Offset
LayerRecordQualifier	Qualifies on the Layer Record Qualifier
EthernetMulticast	Qualify on Ethernet packets on which the destination MAC address is multicast (MAC-DA[40] is set)
EthernetBroadcast	Qualify on Ethernet packets on which the destination MAC address is broadcast (MAC-DA[47:0] is all 1)
EthernetFirstTpidExist	Qualify on Ethernet packets on which their first VLAN TPID exists
EthernetFirstTpidIndex	Qualify on Ethernet packets first VLAN TPID Index
EthernetSecondTpidExist	Qualify on Ethernet packets on which their second VLAN TPID exists
EthernetSecondTpidIndex	Qualify on Ethernet packets second VLAN TPID Index
EthernetThirdTpidExist	Qualify on Ethernet packets on which their third VLAN TPID exists
EthernetThirdTpidIndex	Qualify on Ethernet packets' third VLAN TPID Index
L2Format	Qualify on the type of the frame (see bcmFieldL2FormatXXX)
IpHasOptions	Qualify on IPv4 packets where the IP header includes options
IpFirstFrag	Qualify on IPv4 packets where the IP header is fragmented and this is the first fragment
IpTunnelType	Qualify on IPv4 TunnelType (see bcmFieldTunnelXXX)
Ip6MulticastCompatible	Qualify on IPv6 packets for which their DIP is multicast (8 MSBs are 0xFF)
Ip6FirstAdditionalHeaderExist	Qualify on IPv6 packets where the first additional header exists
Ip6FirstAdditionalHeader	Qualify on IPv6 packets' first additional header (see bcmFieldIp6AdditionalHeaderXXX)
Ip6SecondAdditionalHeaderExist	Qualify on IPv6 packets' second additional header exists
Ip6SecondAdditionalHeader	Qualify on IPv6 packets' second additional header (see bcmFieldIp6AdditionalHeaderXXX)
ItmhPphType	Qualify on Ingress TM header PPH type
LayerRecordTypeRaw	Qualifies on the Layer Record Type. For possible values, see bcm_field_layer_type_t; input expects raw value
Ip4DstMulticast	Qualify on IPv4 packets for which their DIP is multicast
IpTunnelTypeRaw	Raw qualify on IPv4 TunnelType

11.32.2.3 Metadata Qualifiers List

Table 34: List of iPMF-1,2 Qualifiers

bcmFieldQualify…	Description
InPort	Ingress Port, value should be encoded as LOCAL GPORT Type. Includes core ID, and can be used by entries for all cores
L4PortRangeCheck	Qualify on L4 Ops (L4 source and dest ports) range. The qualifier is a bitmap and not specific range ID, it means if range 2 was configured and the qualifier is hit, the entry value is expected to be 0x100.
SrcPort	Qualify on source port, value should be encoded as SYS_PORT GPORT Type
DstPort	Qualify on destination port, all ingress stages expect raw value, egress expects GPORT encoded as SYS_PORT
PacketRes	When TRUE packet lookup result was Unknown Destination
InterfaceClassL2	VSI Profile
Vrf	VRF Id
VlanFormat	Incoming vlan structure format (untagged/single/double/priority-tagged). For possible values, see bcm_port_tag_struct_type_t.
DstMulticastGroup	Multicast Group id, encoded as GPORT
SrcModPortGport	Qualify on Source module/port pair, value should be encoded as SYS_PORT GPORT Type

Table 34: List of iPMF-1,2 Qualifiers (Continued)

bcmFieldQualify…	Description
Color	Qualify based on packet color, value should be one of BCM_FIELD_COLOR_XXX
IntPriority	Internal priority (TC)
Vpn	VSI Id
MplsForwardingLabelAction	MPLS forwarding label action
L2Learn	L2 learn enable
EcnValue	Qualify on metadata ECN or congestion information (including CNI)
RxTrapCode	Qualify on RX Trap Code
Ptch	Opaque attribute field of the Injected packets. Part of the PTCH_2 header.
RxTrapData	Qualify on RX Trap qualifier
TrillEgressRbridge	Egress RBridge Nickname
ISid	I-SID (MAC-in-MAC lookup-id)
DstRpfGport	RPF destination (gport) for the RPF check
TrunkHashResult	Trunk Hash Result (that is, the Load-balancing Key)
PacketLengthRangeCheck	Packet length range, range ID on which the packet matched. If packet size > 144B, it is treated as 144B.
LearnSrcMac	Qualifies on learn source MAC
LearnVlan	Qualifies on learn VLAN
PacketSize	Packet size in Bytes
TranslatedOuterVlanId	Translated Outer VLAN ID
TranslatedOuterVlanPri	Translated Outer VLAN priority
TranslatedOuterVlanCfi	Translated Outer VLAN CFI
TranslatedInnerVlanId	Translated inner VLAN Id
TranslatedInnerVlanPri	Translated inner VLAN priority
TranslatedInnerVlanCfi	Translated inner VLAN CFI
RxTrapCodeForSnoop	Qualify on Snoop Code
IncomingIpIfClass	RIF profile
RxTrapStrength	Qualify on RX Trap Strength
OamUpMep	It indicates if the OAM packet is vUP-MEP (sent to a destination in the network, as opposed to a specific port)
OamSubtype	In OAM the packet type is specified in the OAM header and mapped to a subtype in the hardware
OamHeaderOffset	Indicates the offset of the OAM header relative to the start of packet (as opposed to start of header offset)
OamStampOffset	Indicates the offset to the position, in the OAM header, where the ToD or counter value should be stamped relative to the start of packet
OamMepId	If MEP is handled in OAMP, then the OAM-ID is the MEP-ID (equivalent to the index used to access the MEP DB)
OamMeterDisable	Attribute that is passed to the PMF and can also be configured by the user per MEP
VlanAction	VLAN translation action ID
GeneratedTtl	Qualify on Packet's generate Time-To-Live
IpMulticastCompatible	Packet is compatible for multicast
PacketIsIEEE1588	Indicating whether the packet is 1588
IEEE1588Encapsulation	IEEE-1588 Encapsulation according to bcm_field_IEEE1588Encap_t
IEEE1588CompensateTime Stamp	IEEE-1588 update timestamp
IEEE1588Command	Command used by egress pipeline, indicating if CF(correction field) needs to be updated

Table 34: List of iPMF-1,2 Qualifiers (Continued)

bcmFieldQualify…	Description
IEEE1588HeaderOffset	This field indicates the offset in bytes of the IEEE-1588 header
KeyGenVar	Match on configured key Gen Variable (values that was configured for context)
NetworkQos	Qualify Upon Network QoS
EcmpLoadBalanceKey0	ECMP Load Balance Key 0
EcmpLoadBalanceKey1	ECMP Load Balance Key 1
EcmpLoadBalanceKey2	ECMP Load Balance Key 2
ForwardingLayerIndex	Qualify on the Forwarding Layer Index
AcInLifWideData	Qualifies on the Ac InLIF wide (generic) data
NativeAcInLifWideData	Qualifies on the native Ac InLIF wide (generic) data
TracePacket	Qualifies upon Packets that Trace was set for them
InInterface	Ingress Logical Interface (LIF/RIF), value should be encoded as L3 interface
OutInterface	Egress Logical Interface (LIF/RIF), should be encoded as L3 interface
PortClassPacketProcessing	PP port profile
PortClassTrafficManagement	TM port profile
InVPort0	In LIF 0, value should be GPORT with subtype LIF
InVPort1	In LIF 1, value should be GPORT with subtype LIF
OutVPort0	OutLIF 0, value should be GPORT with subtype LIF
OutVPort1	OutLIF 1, value should be GPORT with subtype LIF
BierStringOffset	Qualifies upon Bier string offset
BierStringSize	Qualifies upon Bier string size
PacketIsBier	Qualifies upon Bier packets
PortClassPacketProcessing GeneralData	PP port general data
ForwardingAdditionalInfo	Qualifies on the Forwarding additional info
InVportClass0	InLIF Profile 0, by Gport
InVportClass1	InLIF Profile 1, by Gport
StatId0	Qualifies on statistics ID 0
StatId1	Qualifies on statistics ID 1
StatId2	Qualifies on statistics ID 2
StatId3	Qualifies on statistics ID 3
StatId4	Qualifies on statistics ID 4
StatId5	Qualifies on statistics ID 5
StatId6	Qualifies on statistics ID 6
StatId7	Qualifies on statistics ID 7
StatId8	Qualifies on statistics ID 8
StatId9	Qualifies on statistics ID 9
StatProfile0	Qualifies on statistics profile 0
StatProfile1	Qualifies on statistics profile 1
StatProfile2	Qualifies on statistics profile 2
StatProfile3	Qualifies on statistics profile 3
StatProfile4	Qualifies on statistics profile 4
StatProfile5	Qualifies on statistics profile 5
StatProfile6	Qualifies on statistics profile 6

Table 34: List of iPMF-1,2 Qualifiers (Continued)

bcmFieldQualify…	Description
StatProfile7	Qualifies on statistics profile 7
StatProfile8	Qualifies on statistics profile 8
StatProfile9	Qualifies on statistics profile 9
StatMetaData	Qualifies on Statistics metadata
InVPort0Raw	Qualifies on In LIF 0 input expects raw value
InVPort1Raw	Qualifies on In LIF 1 input expects raw value
OutVPort0Raw	Qualifies on OutLIF 0 input expects raw value
OutVPort1Raw	Qualifies on OutLIF 1 input expects raw value
MirrorCode	Matches on Mirror Code
LearnVsi	Qualifies on learn VSI
LearnData	Qualifies on data field in the Learn Info
LearnStationMove	Qualifies on learn Station move
LearnMatch	Qualifies on learn Match
LearnFound	Qualifies on learn Found
LearnExpectedWon	Qualifies on learn Expected Won
VrfValue	Qualifies on the VRF part of forwarding data, without checking that the type is VRF
InPortWithoutCore	Ingress Port, value should be encoded as LOCAL GPORT Type. Does not include the core ID, caution should be taken when not adding entry by core
SaLookupAcceptedStrength	Qualifies on Accepted strength LSB returned from SA Lookup result. Qualifier is only relevant for bridged packets
SrcPortRaw	Qualifies on source port, input expects raw value
InPortWithoutCoreRaw	Qualifies on Ingress Port. Does not include the core ID, caution should be taken when not adding entry by core, input expects raw value
VlanFormatRaw	Qualifies on the incoming VLAN structure format (untagged, single tagged, double-tagged, or priority-tagged). The input expects a raw value
InPortRaw	Qualifies on Ingress Port. Includes core ID, and can be used by entries for all cores, input expects raw value
ColorRaw	Qualifies on packet color, input expects raw value
DstRpfValid	Checks if the RPF Destination is valid
RpfOutVPort	Qualifies on the RPF OUT LIF. Value should be GPORT with subtype LIF
RpfOutInterface	Qualifies on the RPF OUT LIF. Value should be L3 interface
RpfOutVPortRaw	Qualifies on the RPF OUT LIF
RpfRouteValid	Qualifies on the RPF Route valid
VipValid	Qualifies on SLLB Virtual Wire VW_VIP_VALID
VipId	Qualifies on SLLB Virtual Wire VW_VIP_ID
VIPMemberReference	Qualifies on SLLB Virtual Wire VW_MEMBER_REFERENCE
PccHit	Qualifies on SLLB Virtual Wire VW_PCC_HIT
PrtQualifier	Qualifies on the prt_qualifier signal, which may be set by the injected packet (Opaque attribute field part of the PTCH_2 header). The input expects an enum value `bcm_field_prt_qualifier_t`. Similar to the ptch qualifier, which uses a raw value and not an enum.

Table 35: List of iPMF-3 Qualifiers

bcmFieldQualify…	Description
InPort	Ingress Port, value should be encoded as LOCAL GPORT Type. Includes core ID, and can be used by entries for all cores
SrcPort	Qualify on source port, value should be encoded as SYS_PORT GPORT Type
DstPort	Qualify on destination port, all ingress stages expect raw value, egress expects GPORT encoded as SYS_PORT
InterfaceClassL2	VSI Profile
Vrf	VRF Id
SrcModPortGport	Qualify on Source module/port pair, value should be encoded as SYS_PORT GPORT Type
Color	Qualify based on packet color, value should be one of BCM_FIELD_COLOR_XXX
IntPriority	Internal priority (TC)
Vpn	VSI Id
FlowId	Qualify on destination Flow-Id
MplsForwardingLabelAction	MPLS forwarding label action
L2Learn	L2 learn enable
EcnValue	Qualify on metadata ECN or congestion information (including CNI)
RxTrapCode	Qualify on RX Trap Code
RxTrapData	Qualify on RX Trap qualifier
TrillEgressRbridge	Egress RBridge Nickname
ISid	I-SID (MAC-in-MAC lookup-id)
DstRpfGport	RPF destination (gport) for the RPF check
TrunkHashResult	Trunk Hash Result (i.e., the Load-balancing Key)
LearnSrcMac	Qualifies on learn source MAC
LearnVlan	Qualifies on learn VLAN
TranslatedOuterVlanId	Translated Outer VLAN ID
TranslatedOuterVlanPri	Translated Outer VLAN priority
TranslatedOuterVlanCfi	Translated Outer VLAN CFI
TranslatedInnerVlanId	Translated inner VLAN Id
TranslatedInnerVlanPri	Translated inner VLAN priority
TranslatedInnerVlanCfi	Translated inner VLAN CFI
DstGport	Qualifies packet according to destination encoded as trap code
RxTrapCodeForSnoop	Qualify on Snoop Code
IncomingIpIfClass	RIF profile
StackingRoute	Stacking Route History bitmap
RxTrapStrength	Qualify on RX Trap Strength
OamUpMep	Indicates if the OAM packet is vUP-MEP (sent to a destination in the network, as opposed to a specific port)
OamSubtype	In OAM the packet type is specified in the OAM header and mapped to a subtype in the hardware
OamHeaderOffset	Indicates the offset of the OAM header relative to the start of packet (as opposed to start of header-offset)
OamStampOffset	Indicates the offset to the position, in the OAM header, where the ToD or counter value should be stamped relative to the start of packet
OamMepId	If MEP is handled in OAMP, then the OAM-ID is the MEP-ID (equivalent to the index used to access the MEP DB)
VlanAction	VLAN translation action ID

Table 35: List of iPMF-3 Qualifiers (Continued)

bcmFieldQualify…	Description
GeneratedTtl	Qualify on Packet's generate Time-To-Live
PacketIsIEEE1588	Indicating whether the packet is 1588
IEEE1588Encapsulation	IEEE-1588 Encapsulation according to bcm_field_IEEE1588Encap_t
IEEE1588CompensateTimeStamp	IEEE-1588 update time stamp
IEEE1588Command	Command used by egress pipeline, indicating if CF (correction field) needs to be update
IEEE1588HeaderOffset	This field indicates the offset in bytes of the IEEE-1588 header
KeyGenVar	Match on configured key Gen Variable (values that was configured for context)
Container	This qualifier will be used as container in IPMF3, to receive the action buffer, when performing cascading between IPMF1/2 and IPMF3
UDHBase0	User-defined Header 0 encoded type and size – Value 1: type PMF size 32b
UDHBase1	User-defined Header 1 encoded type and size – Value 1: type PMF size 32b
UDHBase2	User-defined Header 2 encoded type and size – Value 1: type PMF size 32b
UDHBase3	User-defined Header 3 encoded type and size – Value 1: type PMF size 32b
NetworkQos	Qualify Upon Network QoS
ForwardingLayerIndex	Qualify on the forwarding layer index
IPTProfile	Qualifies upon IPT Profile (End Of Packet Editing)
UDHData0	User-defined Header 0 payload MS bits of UDH
UDHData1	User-defined Header 1 payload
UDHData2	User-defined Header 2 payload
UDHData3	User-defined Header 3 payload LS bits of UDH
RxSnoopStrength	Snoop Strength
StatSamplingCode	Statistical Sampling Code value
StatSamplingQualifier	Statistical Sampling qualifier value
RpfEcmpMode	RPF ECMP mode
StatOamLM	OAM LM counter value, the index of the values are bcmFieldStatOamLmIndexXXX
InInterface	Ingress Logical Interface (LIF/RIF), value should be encoded as L3 interface
OutInterface	Egress Logical Interface (LIF/RIF), should be encoded as L3 interface
PortClassPacketProcessing	PP port profile
PortClassTrafficManagement	TM port profile
InVPort0	In LIF 0, value should be GPORT with subtype LIF
InVPort1	In LIF 1, value should be GPORT with subtype LIF
OutVPort0	OutLIF 0, value should be GPORT with subtype LIF
OutVPort1	OutLIF 1, value should be GPORT with subtype LIF
OutVPort2	OutLIF 2, value should be GPORT with subtype LIF
OutVPort3	OutLIF 3, value should be GPORT with subtype LIF
BierStringOffset	Qualifies upon Bier string offset
BierStringSize	Qualifies upon Bier string size
PacketIsBier	Qualifies upon Bier packets
PortClassPacketProcessingGeneralData	PP port general data
ForwardingAdditionalInfo	Qualifies on the Forwarding additional info
InVportClass0	InLIF Profile 0, by Gport

Table 35: List of iPMF-3 Qualifiers (Continued)

bcmFieldQualify…	Description
InVportClass1	InLIF Profile 1, by Gport
StatMetaData	Qualifies on Statistics metadata
InVPort0Raw	Qualifies on In LIF 0 input expects raw value
InVPort1Raw	Qualifies on In LIF 1 input expects raw value
OutVPort0Raw	Qualifies on OutLIF 0 input expects raw value
OutVPort1Raw	Qualifies on OutLIF 1 input expects raw value
OutVPort2Raw	Qualifies on OutLIF 2 input expects raw value
OutVPort3Raw	Qualifies on OutLIF 3 input expects raw value
MirrorCode	Matches on Mirror Code
MirrorData	Matches on Mirror Data (qualifier)
LearnVsi	Qualifies on learn VSI
LearnData	Qualifies on data field in the Learn Info
LearnStationMove	Qualifies on learn Station move
LearnMatch	Qualifies on learn Match
LearnFound	Qualifies on learn Found
LearnExpectedWon	Qualifies on learn Expected Won
VrfValue	Qualifies on the VRF part of forwarding data, without checking that the type is VRF
InPortWithoutCore	Ingress Port, value should be encoded as LOCAL GPORT Type. Does not include the core ID, caution should be taken when not adding entry by core
SrcPortRaw	Qualifies on source port, input expects raw value
InPortWithoutCoreRaw	Qualifies on Ingress Port. Does not include the core ID, caution should be taken when not adding entry by core, input expects raw value
InPortRaw	Qualifies on Ingress Port. Includes core ID, and can be used by entries for all cores, input expects raw value
ColorRaw	Qualifies on packet color, input expects raw value
DstRpfValid	Checks if the RPF Destination is valid
EcmpGroup	Qualifies on the ECMP group
StatOamLMRaw	OAM LM counter raw value

Table 36: List of Egress-PMF Qualifiers

bcmFieldQualify…	Description
L4PortRangeCheck	Qualify on L4 Ops (L4 source and dest ports) range. The qualifier is a bitmap and not specific range id, it means if range 2 was configured and the qualifier is hit, the entry value is expected to be 0x100
SrcPort	Qualify on source port, value should be encoded as SYS_PORT GPORT Type
DstPort	Qualify on destination port, all ingress stages expect raw value, egress expects GPORT encoded as SYS_PORT
SrcClassField	PPH Value 2
DstClassField	PPH Value 1
InterfaceClassL2	VSI Profile
Vrf	VRF Id
OutPort	Egress port, value should be encoded as LOCAL GPORT Type. Does not include the core ID, caution should be taken when not adding entry by core
DstL3Egress	L3 Egress Interface, Raw value of SYS_PORT

Table 36: List of Egress-PMF Qualifiers (Continued)

bcmFieldQualify…	Description
DstMulticastGroup	Multicast Group id, encoded as GPORT
SrcModPortGport	Qualify on Source module/port pair, value should be encoded as SYS_PORT GPORT Type
Color	Qualify based on packet color, value should be one of BCM_FIELD_COLOR_XXX
IntPriority	Internal priority (TC)
Vpn	VSI Id
RepCopy	Qualify on Copy type: 0:Forwarded / 1:Snooped / 2:Mirrored / 3:Statistical Sampling, packet
EcnValue	Qualify on metadata ECN or congestion information (including CNI)
RxTrapCode	Qualify on RX Trap Code
TrunkHashResult	Trunk Hash Result (i.e., the Load-balancing Key)
LearnSrcMac	Qualifies on learn source MAC
LearnVlan	Qualifies on learn VLAN
Fhei	DNX FHEI header field
FheiSize	DNX FHEI header size in bytes
StackingRoute	Stacking Route History bitmap
RxTrapStrength	Qualify on RX Trap Strength
OamTsSystemHeader	FTMH Application specific extension
DstSysPortExt	Qualifies upon TM Destination extension header
GeneratedTtl	Qualify on Packet's generate Time-To-Live
PphPresent	Qualifies on PPH present field in ITMH
PacketProcessingInVportClass	Qualifies on InLIF profile
UDHBase0	User-defined Header 0 encoded type and size – Value 1: type PMF size 32b
UDHBase1	User-defined Header 1 encoded type and size – Value 1: type PMF size 32b
UDHBase2	User-defined Header 2 encoded type and size – Value 1: type PMF size 32b
UDHBase3	User-defined Header 3 encoded type and size – Value 1: type PMF size 32b
NetworkQos	Qualify Upon Network Qos
AceEntryId	Qualify Upon Ace Entry ID
NetworkLoadBalanceKey	Network Load Balance Key
IPTProfile	Qualifies upon IPT Profile (End Of Packet Editing)
UDHData0	User-defined Header 0 payload MS bits of UDH
UDHData1	User-defined Header 1 payload
UDHData2	User-defined Header 2 payload
UDHData3	User-defined Header 3 payload LS bits of UDH
RxSnoopStrength	Snoop Strength
RxSnoopCode	qualify upon Snoop profile
InInterface	Ingress Logical Interface (LIF/RIF), value should be encoded as L3 interface
OutInterface	Egress Logical Interface (LIF/RIF), should be encoded as L3 interface
OutVportClass	OutLIF Profile, by Gport
PortClassPacketProcessing	PP port profile
PortClassTrafficManagement	TM port profile
OutPortTrafficManagement	Out TM-port
InVPort0	In LIF 0, value should be GPORT with subtype LIF
OutVPort0	OutLIF 0, value should be GPORT with subtype LIF
OutVPort1	OutLIF 1, value should be GPORT with subtype LIF

Table 36: List of Egress-PMF Qualifiers (Continued)

bcmFieldQualify…	Description
OutVPort2	OutLIF 2, value should be GPORT with subtype LIF
OutVPort3	OutLIF 3, value should be GPORT with subtype LIF
ForwardingAdditionalInfo	Qualifies on the Forwarding additional info
InVportClass0	InLIF Profile 0, by Gport
InVPort0Raw	Qualifies on In LIF 0 input expects raw value
OutVPort0Raw	Qualifies on OutLIF 0 input expects raw value
OutVPort1Raw	Qualifies on OutLIF 1 input expects raw value
OutVPort2Raw	Qualifies on OutLIF 2 input expects raw value
OutVPort3Raw	Qualifies on OutLIF 3 input expects raw value
LearnVsi	Qualifies on learn VSI
LearnData	Qualifies on data field in the Learn Info
VrfValue	Qualifies on the VRF part of forwarding data, without checking that the type is VRF
DstSysPortExtPresent	Qualifies upon if TM Destination extension header is present
SrcPortRaw	Qualifies on source port, input expects raw value
ColorRaw	Qualifies on packet color, input expects raw value
DstPortRaw	Qualifies on destination port, input expects raw value
OutPortRaw	Qualifies on Egress port. Does not include the core ID, caution should be taken when not adding entry by core, input expects raw value
McastPkt	All ingress replicas will have this value set to 1
LearnExtensionPresent	Qualifies on whether PPH learn extension is present
FtmhAsePresent	Qualifies packet on FTMH ASE present field

NOTE: Qualifier sizes can be obtained by using the following shell command:

```
field map qualifier bcm <stage=ipmf1/2/3..epmf>
```

11.32.3 Actions List

To see latest version support, run the following diag command:

```
field action Predefined stage=iPMF1/2/3/ePMF
```

Table 37: iPMF-1,2 Actions

bcmFieldAction…	Description
PrioIntNew	TC update
EcnNew	Change metadata ECN or congestion information (including CNI)
Redirect	Redirect packet to destination
RedirectMcast	Redirect to Multicast Group Id
MirrorIngress	Set mirror, value should be encoded as GPORT Type Mirror for mirror command
L3Switch	Sets packet destination, input expects L3 interface
DropPrecedence	Set Packet's Drop Precedence, the values are BCM_FIELD_COLOR_XXX
VrfSet	Set VRF. value[0]: fwd_domain_id_type. value[1]: fwd_domain_id_value. value[1]: fwd_domain_profile. In iPMF3, fwd_domain_profile isn't updated
InnerVlanNew	Replace inner VLAN Id
OuterVlanNew	Replace outer VLAN Id

Table 37: iPMF-1,2 Actions (Continued)

bcmFieldAction…	Description
OuterVlanPrioNew	Replace outer VLAN tag priority
VSQ	Assign matching packets to specified VSQ
Trap	Set Trap code, use TRAP GPORT type value[0]=TRAP_GPORT value[1]=Trap Qualifier. at egress stage refers only to the TRAP ID (strength is not relevant, and can be changed using bcmFieldActionTrapStrength action)
UsePolicerResult	Specify/override where policer result will be used for matched packets
Snoop	Snoop matched packets, input can be either a snoop command: value[0] = Mirror Gport, value[1]=snoop qualifier, value[2]=snoop strength OR: snoop code: value[0] = Trap Gport, value[1]=snoop qualifier, at egress stage refers only to the snoop ID (strength is not relevant, and can be changed using bcmFieldActionSnoopStrength action)
TtlSet	New TTL
DstRpfGportNew	Set the RPF Destination
SrcGportNew	Set the Source-Port
SystemHeaderSet	Modify EEI
VSwitchNew	New VSI. value[0]: fwd_domain_id_type. value[1]: fwd_domain_id_value. value[1]: fwd_domain_profile. In iPMF3, fwd_domain_profile is not updated
VlanActionSetNew	Modify the VLAN Action Set Id
IngressGportSet	Set a new ingress Global Lif encoded, Values should be Encoded as GPORT
StackingRouteNew	Replace the value of the stacking route
PphPresentSet	If set, a packet processing header is present
FabricHeaderSet	Change System Header Profile, bcm_field_system_header_profile_t
Oam	Changes 4 OAM signals. value[0]: OAM-Stamp-Offset. value[1]: OAM-offset. value[2]: OAM-Sub-Type. value[3]: OAM-Up-Mep
PacketTraceEnable	Enable Trace packet
IEEE1588	Setting various parameters for 1588 frames. Bits(0:7): header_offset. Bit(8): encapsulation. Bits(9:10) command. Bit(11): compensate_time_stamp. Bit(12): packet_is_ieee1588
Forward	Forward Destination Raw value, setting to 1's will drop the packet
PphSnoopCode	Set the PPH Snoop code value (Reserved Bits)
IngressTimeStampInsert	Insert IPIPE time stamp
AdmitProfile	Admit Profile
LatencyFlowId	Change the latency flow-Id. value[0] valid. value[1]: latency_flow_id. value[2]: latency_flow_profile
VisibilityEnable	Enables visibility for current packet
InVportClass0	Updates the profile of an in LIF encoded as Gport
InVportClass1	Updates the profile of an in LIF encoded as Gport
NetworkQos	Updates Network Qos value
Void	Void action, used for cascading, saving buffer in the action payload (for FEM use)
Container	This action will be used as container in IPMF2, to parse the action buffer, when performing cascading between IPMF2 and IPMF3
EgressForwardingIndex	Indicate for egress parser which layer is the Forwarding layer
UDHData0	Set the Payload for User-defined Header 0
UDHData1	Set the Payload for User-defined Header 1
UDHData2	Set the Payload for User-defined Header 2
UDHData3	Set the Payload for User-defined Header 3
UDHBase0	Set the Base for User-defined Header 0
UDHBase1	Set the Base for User-defined Header 1

Table 37: iPMF-1,2 Actions (Continued)

bcmFieldAction…	Description
UDHBase2	Set the Base for User-defined Header 2
UDHBase3	Set the Base for User-defined Header 3
IPTProfile	IPT Profile action (used for instrumentation)
StatId0	Updates statistics ID 0
StatId1	Updates statistics ID 1
StatId2	Updates statistics ID 2
StatId3	Updates statistics ID 3
StatId4	Updates statistics ID 4
StatId5	Updates statistics ID 5
StatId6	Updates statistics ID 6
StatId7	Updates statistics ID 7
StatId8	Updates statistics ID 8
StatId9	Updates statistics ID 9
StatProfile0	Updates statistics profile 0. value[0]: is_meter. value[1]: is_lm. value[2]: type. value[3]: valid
StatProfile1	Updates statistics profile 1. value[0]: is_meter. value[1]: is_lm. value[2]: type. value[3]: valid
StatProfile2	Updates statistics profile 2. value[0]: is_meter. value[1]: is_lm. value[2]: type. value[3]: valid
StatProfile3	Updates statistics profile 3. value[0]: is_meter. value[1]: is_lm. value[2]: type. value[3]: valid
StatProfile4	Updates statistics profile 4. value[0]: is_meter. value[1]: is_lm. value[2]: type. value[3]: valid
StatProfile5	Updates statistics profile 5. value[0]: is_meter. value[1]: is_lm. value[2]: type. value[3]: valid
StatProfile6	Updates statistics profile 6. value[0]: is_meter. value[1]: is_lm. value[2]: type. value[3]: valid
StatProfile7	Updates statistics profile 7. value[0]: is_meter. value[1]: is_lm. value[2]: type. value[3]: valid
StatProfile8	Updates statistics profile 8. value[0]: is_meter. value[1]: is_lm. value[2]: type. value[3]: valid
StatProfile9	Updates statistics profile 9. value[0]: is_meter. value[1]: is_lm. value[2]: type. value[3]: valid
InvalidNext	Invalid next Action macro Id
IngressDoNotLearn	Disable Ingress learning
EgressDoNotLearn	Disable Egress learning
VisibilityClear	Disables visibility for current packet
StatMetaData	Set statistics metaData
InVport0	Change the InLIF 0, value should be encoded as a GPORT with LIF subtype
InVport1	Change the InLIF 1, value should be encoded as a GPORT with LIF subtype
OutVport0	Change OutLIF 0, value should be encoded as a GPORT with LIF subtype
OutVport1	Change OutLIF 1, value should be encoded as a GPORT with LIF subtype
OutVport2	Change OutLIF 2, value should be encoded as a GPORT with LIF subtype
OutVport3	Change OutLIF 3, value should be encoded as a GPORT with LIF subtype
ForwardingTypeMeterMap	Maps L2 forwarding type to one of the general meters
StatSampling	Update statistical sampling value, value encoded as Mirror Gport
StatOamLM	Update the counter of OAM LM, the index of the values are bcmFieldStatOamLmIndexXXX
ForwardingLayerIndex	Update forwarding layer index
InInterface0	Change the InLIF 0, value should be encoded as an interface
InInterface1	Change the InLIF 1, value should be encoded as an interface
OutInterface0	Change OutLIF 0, value should be encoded as type interface
OutInterface1	Change OutLIF 1, value should be encoded as type interface
OutInterface2	Change OutLIF 2, value should be encoded as type interface

Table 37: iPMF-1,2 Actions (Continued)

bcmFieldAction…	Description
OutInterface3	Change OutLIF 3, value should be encoded as type interface
BierStringOffset	Updates Bier string offset
BierStringSize	Updates Bier string size
PacketIsBier	Updates is Bier packet
Eventor	Enable push data to eventor
ForwardingAdditionalInfo	Updates forwarding additional information
SmallExemLearn	Updates the Small Exact Match learn info (IPMF2 only)
InVport0Raw	Change the InLIF 0, with raw value
InVport1Raw	Change the InLIF 1, with raw value
OutVport0Raw	Change the OutLIF 0, with raw value
OutVport1Raw	Change the OutLIF 1, with raw value
OutVport2Raw	Change the OutLIF 2, with raw value
OutVport3Raw	Change the OutLIF 3, with raw value
IngressTimeStampInsertValid	Insert ingress timestamp. Overrides the time stamp in TSH header
SnoopRaw	Snoop matched packets, input should be raw value. At egress stage refers only to the snoop ID (strength is not relevant, and can be changed using bcmFieldActionSnoopStrength action)
DropPrecedenceRaw	Set Packet's Drop Precedence, input expects raw value
FabricHeaderSetRaw	Change System Header Profile, input expects raw value
SrcGportNewRaw	Set the Source-Port, input expects raw value
LatencyFlowIdRaw	Change the latency flow-Id, input expects raw value
MirrorIngressRaw	Set mirror in ingress, input expects raw value
OamRaw	Changes 4 OAM signals (OAM-Stamp-Offset, OAM-offset, OAM-Sub-Type, OAM-Up-Mep), input expects raw value
TrapRaw	Set Trap code, input should be raw value. At egress stage refers only to the TRAP ID (strength is not relevant, and can be changed using bcmFieldActionTrapStrength action)
VrfSetRaw	Set VRF, input expects raw value
StatProfile0Raw	Updates statistics profile 0. Input expects raw value
StatProfile1Raw	Updates statistics profile 1. Input expects raw value
StatProfile2Raw	Updates statistics profile 2. Input expects raw value
StatProfile3Raw	Updates statistics profile 3. Input expects raw value
StatProfile4Raw	Updates statistics profile 4. Input expects raw value
StatProfile5Raw	Updates statistics profile 5. Input expects raw value
StatProfile6Raw	Updates statistics profile 6. Input expects raw value
StatProfile7Raw	Updates statistics profile 7. Input expects raw value
StatProfile8Raw	Updates statistics profile 8. Input expects raw value
StatProfile9Raw	Updates statistics profile 9. Input expects raw value
StatSamplingRaw	Update statistical sampling value, input expects raw value
IpMulticastCompatible	Designates when a Compatible MC procedure found a match
RpfOutVport	Updates the RPF OUT LIF. value[0]: route_valid, value[1]: default_route_found, value[2]: rpf_out_lif (encoded as GPORT with LIF subtype)
RpfOutInterface	Updates the RPF OUT LIF. value[0]: route_valid, value[1]: default_route_found, value[2]: rpf_out_lif (encoded as L3 interface)
RpfOutVportRaw	Updates the RPF OUT LIF. Input expects raw value
LearnRaw0	Updates learn info 0

Table 37: iPMF-1,2 Actions (Continued)

bcmFieldAction…	Description
LearnRaw1	Updates learn info 1
LearnRaw2	Updates learn info 2
LearnRaw3	Updates learn info 3
LearnRaw4	Updates learn info 4
DstRpfGportNewValid	Set the RPF Destination Valid
DstRpfGportNewRaw	Set the RPF Destination. Input expects raw value
StatOamLMRaw	Update the counter of OAM LM with a raw value
LearnKey0	Updates bits 31:0 of learn info key
LearnKey1	Updates bits 63:32 of learn info key
LearnKey2	Updates learn info key and app_db_index. value[0]: bits 79:64 of learn info ley, value[1] app_db_index
LearnKey2Raw	Updates learn info key and app_db_index. Key is at the LSB
LearnPayload0	Updates bits 31:0 of learn info data (payload)
LearnPayload1	Updates bits 63:32 of learn info data (payload)
LearnEntryFormatIndex	Updates entry format index in learn info
LearnOrTransplant	Updates learn or transplant bit
NetworkLoadBalanceKey	Set the network load balancing Key

Table 38: iPMF-3 Actions

bcmFieldAction…	Description
PrioIntNew	TC update
EcnNew	Change metadata ECN or congestion information (including CNI)
Redirect	Redirect packet to to destination
RedirectMcast	Redirect to Multicast Group Id
MirrorIngress	set mirror, value should be encoded as GPORT Type Mirror for mirror command
L3Switch	Sets packet destination, input expects L3 interface
DropPrecedence	Set Packet's Drop Precedence, the values are BCM_FIELD_COLOR_XXX
VrfSet	Set VRF. value[0]: fwd_domain_id_type. value[1]: fwd_domain_id_value. value[1]: fwd_domain_profile. In iPMF3, fwd_domain_profile isn't updated
InnerVlanNew	Replace inner VLAN Id
OuterVlanNew	Replace outer VLAN Id
OuterVlanPrioNew	Replace outer VLAN tag priority
VSQ	Assign matching packets to specified VSQ
Trap	Set Trap code, use TRAP GPORT type value[0]=TRAP_GPORT value[1]=Trap Qualifier. at egress stage refers only to the TRAP ID (strength is not relevant, and can be changed using bcmFieldActionTrapStrength action)
UsePolicerResult	Specify/override where policer result will be used for matched packets
Snoop	Snoop matched packets, input can be either a snoop command: value[0] = Mirror Gport, value[1]=snoop qualifier, value[2]=snoop strength OR: snoop code: value[0] = Trap Gport, value[1]=snoop qualifier, at egress stage refers only to the snoop ID (strength is not relevant, and can be changed using bcmFieldActionSnoopStrength action)
TtlSet	New TTL
TrunkHashKeySet	Set the Trunk Hash Key
SrcGportNew	Set the Source-Port

Table 38: iPMF-3 Actions (Continued)

bcmFieldAction…	Description
StartPacketStrip	Strip the start of packet. value[0]: bcm_field_start_packet_strip_t value[1]: extra bytes to remove
SystemHeaderSet	Modify EEI
VSwitchNew	New VSI. value[0]: fwd_domain_id_type. value[1]: fwd_domain_id_value. value[1]: fwd_domain_profile. In iPMF3, fwd_domain_profile isn't updated
VlanActionSetNew	Modify the VLAN Action Set Id
IngressGportSet	Set a new ingress Global Lif encoded, Values should be Encoded as GPORT
StackingRouteNew	Replace the value of the stacking route
PphPresentSet	If set, a packet processing header is present
FabricHeaderSet	Change System Header Profile, bcm_field_system_header_profile_t
Oam	Changes 4 OAM signals. value[0]: OAM-Stamp-Offset. value[1]: OAM-offset. value[2]: OAM-Sub-Type. value[3]: OAM-Up-Mep
IEEE1588	Setting various parameters for 1588 frames. Bits(0:7): header_offset. Bit(8): encapsulation. Bits(9:10) command. Bit(11): compensate_time_stamp. Bit(12): packet_is_ieee1588
Forward	Forward Destination Raw value, setting to 1's will drop the packet
PphSnoopCode	Set the PPH Snoop code value (Reserved Bits)
AdmitProfile	Admit Profile
LatencyFlowId	Change the latency flow-Id. value[0] valid. value[1]: latency_flow_id. value[2]: latency_flow_profile
VisibilityEnable	Enables visibility for current packet
SystemHeaderSizeAdjust	System Header Size adjust
InVportClass0	Updates the profile of an in LIF encoded as Gport
InVportClass1	Updates the profile of an in LIF encoded as Gport
NetworkQos	Updates Network Qos value
Void	Void action, used for cascading, saving buffer in action payload (for FEM use)
ParsingStartType	Indicates the layer type that egress parser should consider. For information about the possible layer types, see Section 11.30, Common Types.
ParsingStartOffset	Indicate from which offset egress parser should start parsing (all info before that will not be parsed)
EgressForwardingIndex	Indicate for egress parser which layer is the Forwarding layer
UDHData0	Set the Payload for User-defined Header 0
UDHData1	Set the Payload for User-defined Header 1
UDHData2	Set the Payload for User-defined Header 2
UDHData3	Set the Payload for User-defined Header 3
UDHBase	Set the Base for All User-defined Header 0-3, UDHBase0 is at LSB
IPTProfile	IPT Profile action (used for instrumentation)
IPTCommand	IPT Profile command. value[0]: int_commnad. value[1]: int_profile
StatId0	Updates statistics ID 0
StatId1	Updates statistics ID 1
StatId2	Updates statistics ID 2
StatId3	Updates statistics ID 3
StatId4	Updates statistics ID 4
StatId5	Updates statistics ID 5
StatId6	Updates statistics ID 6
StatId7	Updates statistics ID 7
StatId8	Updates statistics ID 8
StatId9	Updates statistics ID 9

Table 38: iPMF-3 Actions (Continued)

bcmFieldAction…	Description
StatProfile0	Updates statistics profile 0. value[0]: is_meter. value[1]: is_lm. value[2]: type. value[3]: valid
StatProfile1	Updates statistics profile 1. value[0]: is_meter. value[1]: is_lm. value[2]: type. value[3]: valid
StatProfile2	Updates statistics profile 2. value[0]: is_meter. value[1]: is_lm. value[2]: type. value[3]: valid
StatProfile3	Updates statistics profile 3. value[0]: is_meter. value[1]: is_lm. value[2]: type. value[3]: valid
StatProfile4	Updates statistics profile 4. value[0]: is_meter. value[1]: is_lm. value[2]: type. value[3]: valid
StatProfile5	Updates statistics profile 5. value[0]: is_meter. value[1]: is_lm. value[2]: type. value[3]: valid
StatProfile6	Updates statistics profile 6. value[0]: is_meter. value[1]: is_lm. value[2]: type. value[3]: valid
StatProfile7	Updates statistics profile 7. value[0]: is_meter. value[1]: is_lm. value[2]: type. value[3]: valid
StatProfile8	Updates statistics profile 8. value[0]: is_meter. value[1]: is_lm. value[2]: type. value[3]: valid
StatProfile9	Updates statistics profile 9. value[0]: is_meter. value[1]: is_lm. value[2]: type. value[3]: valid
InvalidNext	Invalid next Action macro Id
StatMetaData	Set statistics metaData
InVport0	Change the InLIF 0, value should be encoded as a GPORT with LIF subtype
InVport1	Change the InLIF 1, value should be encoded as a GPORT with LIF subtype
OutVport0	Change OutLIF 0, value should be encoded as a GPORT with LIF subtype
OutVport1	Change OutLIF 1, value should be encoded as a GPORT with LIF subtype
OutVport2	Change OutLIF 2, value should be encoded as a GPORT with LIF subtype
OutVport3	Change OutLIF 3, value should be encoded as a GPORT with LIF subtype
ForwardingTypeMeterMap	Maps L2 forwarding type to one of the general meters
StatSampling	Update statistical sampling value, value encoded as Mirror Gport
StatOamLM	Update the counter of OAM LM, the index of the values are bcmFieldStatOamLmIndexXXX
ForwardingLayerIndex	Update forwarding layer index
InInterface0	Change the InLIF 0, value should be encoded as an interface
InInterface1	Change the InLIF 1, value should be encoded as an interface
OutInterface0	Change OutLIF 0, value should be encoded as type interface
OutInterface1	Change OutLIF 1, value should be encoded as type interface
OutInterface2	Change OutLIF 2, value should be encoded as type interface
OutInterface3	Change OutLIF 3, value should be encoded as type interface
BierStringOffset	Updates Bier string offset
BierStringSize	Updates Bier string size
PacketIsBier	Updates is Bier packet
ForwardingAdditionalInfo	Updates forwarding additional information
InVport0Raw	Change the InLIF 0, with raw value
InVport1Raw	Change the InLIF 1, with raw value
OutVport0Raw	Change the OutLIF 0, with raw value
OutVport1Raw	Change the OutLIF 1, with raw value
OutVport2Raw	Change the OutLIF 2, with raw value
OutVport3Raw	Change the OutLIF 3, with raw value
AppendPointerCompensation	Configures pointer to header append compensation value
IngressTimeStampInsertValid	Insert ingress timestamp. Overrides the time stamp in TSH header
SnoopRaw	Snoop matched packets, input should be raw value. At egress stage refers only to the snoop ID (strength is not relevant, and can be changed using bcmFieldActionSnoopStrength action)
DropPrecedenceRaw	Set Packet's Drop Precedence, input expects raw value

Table 38: iPMF-3 Actions (Continued)

bcmFieldAction…	Description
FabricHeaderSetRaw	Change system header profile, input expects raw value
SrcGportNewRaw	Set the Source-Port, input expects raw value
LatencyFlowIdRaw	Change the latency flow-Id, input expects raw value
MirrorIngressRaw	Set mirror in ingress, input expects raw value
OamRaw	Changes 4 OAM signals (OAM-Stamp-Offset, OAM-offset, OAM-Sub-Type, OAM-Up-Mep), input expects raw value
TrapRaw	Set Trap code, input should be raw value. At egress stage refers only to the TRAP ID (strength is not relevant, and can be changed using bcmFieldActionTrapStrength action)
StatProfile0Raw	Updates statistics profile 0. Input expects raw value
StatProfile1Raw	Updates statistics profile 1. Input expects raw value
StatProfile2Raw	Updates statistics profile 2. Input expects raw value
StatProfile3Raw	Updates statistics profile 3. Input expects raw value
StatProfile4Raw	Updates statistics profile 4. Input expects raw value
StatProfile5Raw	Updates statistics profile 5. Input expects raw value
StatProfile6Raw	Updates statistics profile 6. Input expects raw value
StatProfile7Raw	Updates statistics profile 7. Input expects raw value
StatProfile8Raw	Updates statistics profile 8. Input expects raw value
StatProfile9Raw	Updates statistics profile 9. Input expects raw value
StatSamplingRaw	Update statistical sampling value, input expects raw value
StartPacketStripRaw	Strip the start of packet. Refers to bcm_field_start_packet_strip_t. Input expects raw value
ParsingStartTypeRaw	Indicate the layer type that egress parser should consider, input expects raw value
ParsingStartOffsetRaw	Indicate from which offset egress parser should start parsing (all info before that will not be parsed). Input expects raw value
IPTCommandRaw	IPT Profile command. Input expects raw value
LearnRaw0	Updates learn info 0
LearnRaw1	Updates learn info 1
LearnRaw2	Updates learn info 2
LearnRaw3	Updates learn info 3
LearnRaw4	Updates learn info 4
StatOamLMRaw	Update the counter of OAM LM with a raw value
LearnKey0	Updates bits 31:0 of learn info key
LearnKey1	Updates bits 63:32 of learn info key
LearnKey2	Updates learn info key and app_db_index. value[0]: bits 79:64 of learn info ley, value[1] app_db_index
LearnKey2Raw	Updates learn info key and app_db_index. Key is at the LSB
LearnPayload0	Updates bits 31:0 of learn info data (payload)
LearnPayload1	Updates bits 63:32 of learn info data (payload)
LearnEntryFormatIndex	Updates entry format index in learn info
LearnOrTransplant	Updates learn or transplant bit

Table 39: Egress-PMF Actions

bcmFieldAction…	Description
PrioIntNew	TC update **NOTE:** In the ePMF, setting MC traffic to TC=4 is not allowed and can cause traffic issues. Do not use this setting in that case.
Drop	Discard packet
MirrorEgress	Update the mirror profile, mirror_cmd should be encoded as MIRROR Gport
DropPrecedence	Set Packet's Drop Precedence, the values are BCM_FIELD_COLOR_XXX
Trap	Set Trap code, use TRAP GPORT type value[0]=TRAP_GPORT value[1]=Trap Qualifier. at egress stage refers only to the TRAP ID (strength is not relevant, and can be changed using bcmFieldActionTrapStrength action)
Snoop	Snoop matched packets, input can be either a snoop command: value[0] = Mirror Gport, value[1]=snoop qualifier, value[2]=snoop strength OR: snoop code: value[0] = Trap Gport, value[1]=snoop qualifier, at egress stage refers only to the snoop ID (strength is not relevant, and can be changed using bcmFieldActionSnoopStrength action)
QosMapIdNew	Set the QoS profile
Void	Void action, used for cascading, saving buffer in the action payload (for FEM use) or external lookup result
StatId0	Updates statistics ID 0
StatId1	Updates statistics ID 1
StatProfile0	Updates statistics profile 0. value[0]: is_meter. value[1]: is_lm. value[2]: type. value[3]: valid.
StatProfile1	Updates statistics profile 1. value[0]: is_meter. value[1]: is_lm. value[2]: type. value[3]: valid.
InvalidNext	Invalid next Action macro Id
AceEntryId	Pointer to the ACE entry ID
TrapStrength	Updates the Trap Strength. Use TRAP Gport
SnoopStrength	updates the snoop strength
OutPortTrafficManagement	Updates the Out-TM port
SnoopRaw	Snoop matched packets, input should be raw value. At egress stage refers only to the snoop ID (strength is not relevant, and can be changed using bcmFieldActionSnoopStrength action)
DropPrecedenceRaw	Set Packet's Drop Precedence, input expects raw value.
MirrorEgressRaw	Update the egress mirror profile, input expects raw value.
TrapStrengthRaw	Updates the Trap Strength, input expects raw value.
SnoopStrengthRaw	Updates the Snoop Strength, input expects raw value.

Table 40: Egress-Extension (ACE) PMF Actions

bcmFieldAction…	Description
PrioIntNew	TC update
EcnNew	Change metadata ECN or congestion information (including CNI)
DropPrecedence	Set Packet's Drop Precedence, the values are BCM_FIELD_COLOR_XXX
Stat0	Stat meter object. value[0]: stat_meter_obj_cmd. value[1]: stat_meter_obj_id
Stat1	Stat counter object. value[0]: stat_counter_obj_cmd. value[1]: stat_counter_obj_id. value[2]: valid
TtlSet	New TTL
LearnSrcPortNew	Replace the learnt packet source Port
InterfaceClassVPort	Set InLIF profile
Void	Void action, used for cascading, saving buffer in the action payload (for FEM use)

Table 40: Egress-Extension (ACE) PMF Actions (Continued)

bcmFieldAction…	Description
IPTProfile	IPT Profile action (used for instrumentation)
InvalidNext	Invalid next Action macro Id
OutVport0	Change OutLIF 0, value should be encoded as a GPORT with LIF subtype
OutVport1	Change OutLIF 1, value should be encoded as a GPORT with LIF subtype
OutVport2	Change OutLIF 2, value should be encoded as a GPORT with LIF subtype
OutVport3	Change OutLIF 3, value should be encoded as a GPORT with LIF subtype
ForwardingLayerIndex	Update forwarding layer index
OutInterface0	Change OutLIF 0, value should be encoded as type interface
OutInterface1	Change OutLIF 1, value should be encoded as type interface
OutInterface2	Change OutLIF 2, value should be encoded as type interface
OutInterface3	Change OutLIF 3, value should be encoded as type interface
AceContextValue	Updates programmable value per ACE context, bcm_field_ace_context_t.. Up to four different values can be given in value[0-3], although some may be mutually exclusive.
ForwardingAdditionalInfo	Updates forwarding additional information
OutVport0Raw	Change the OutLIF 0, with raw value
OutVport1Raw	Change the OutLIF 1, with raw value
OutVport2Raw	Change the OutLIF 2, with raw value
OutVport3Raw	Change the OutLIF 3, with raw value
DropPrecedenceRaw	Set Packet's Drop Precedence, input expects raw value
AceContextValueRaw	Updates programmable value per ACE context, input expects raw value
Stat0Raw	Stat meter object. Input expects raw value
Stat1Raw	Stat counter object. Input expects raw value

NOTE: Action sizes can be obtained by using the following shell command:

```
field map action bcm <stage=ipmf1/2/3..epmf>
```

11.33 Information Specific to the BCM88800 (Jericho2C) Device

The following table shows the number of various resources per stage.

Resource	Stage			
	iPMF-1	**iPMF-2**	**iPMF-3**	**ePMF**
Context selection lines	128	128	128	128
Contexts	64		64	64
Total number of keys	5+ 5 initial keys	5	3	4
TCAM lookup keys	4	4	2	2
Direct extraction keys	0	2	1	1
Exact match keys	1	1		1
FES contexts	32		16	12
FEM	16		0	0

Table 41: BCM88800 Database Properties

Type	TCAM Lookup
Maximum key size	Possible key sizes
	80b/160b/320b
Maximum payload size	32b/64b/128b
Memory	Shared (with other TCAM FGs)
Maximum number of entries per FG	50K/25K/12.5K
Supported stages	iPMF1/2/3 ePMF
Maximum FGs (in a given stage) per context:	4/4/2 – iPMF1/2 2/2/1 – iPMF3 2/2/1 – ePMF **NOTE:** For ePMF, double lookup key size must be > 160b.

11.33.1 CS Qualifiers

For a list of Context Selection Qualifiers, see Section 11.31.1, CS Qualifiers.

To see latest version support, run the following diagnostic command:

```
field qualifier ContextSelection
```

Table 42: List of iPMF1 Stage Context Selection Qualifiers

bcmFieldQualify…	Description
ForwardingType	Use this qualifier together with `input_arg` to indicate which layer number relative to forwarding layer to qualify upon. Possible layer types are bcmFieldLayerTypeXXX.

Table 42: List of iPMF1 Stage Context Selection Qualifiers (Continued)

bcmFieldQualify…	Description
VlanFormat	Incoming vlan structure format (untagged/single/double/priority-tagged). For possible values, see bcm_port_tag_struct_type_t.
RxTrapCode	Qualify on RX Trap Code
Ptch	Opaque attribute field of the Injected packets. Part of the PTCH_2 header. The input expects a raw value.
VPortRangeCheck	Virtual Port Range Check Results
AppType	Packet Application type used by the ACL lookup. Can be either predefined (bcm_field_AppType_t) or user defined
ForwardingLayerIndex	Qualify on the Forwarding Layer Index
ForwardingLayerQualifier	Use this qualifier together with `input_arg` to indicate which layer number relative to forwarding layer to qualify upon. It qualifies Layer Qualifiers (look for layer record qualifiers in UM).
TracePacket	Qualifies upon Packets that Trace was set for them
InVportClass	Qualify on the profile of an in LIF encoded as Gport
PortClassPacketProcessing	PP port profile
PortClassTrafficManagement	TM port profile
AppTypePredefined	The predefined AppType (`bcm_field_AppType_t`) used by the forwarding lookup
VlanFormatRaw	Qualifies on the incoming VLAN structure format (untagged, single tagged, double-tagged, or priority-tagged). The input expects a raw value.
AppTypeRaw	Qualifies on the packet application type used by the ACL lookup. Can be either predefined or user-defined. The input expects a raw value.
AppTypePredefinedRaw	Qualifies on the predefined AppType used by the forwarding lookup. The input expects a raw value.
ForwardingTypeRaw	The same qualifier as bcmFieldQualifyForwardingType but does not have a mapping function. In other words, the input expects a raw value (HW value).
PrtQualifier	Qualifies on the prt_qualifier signal, which may be set by the injected packet (Opaque attribute field part of the PTCH_2 header). The input expects an enum value `bcm_field_prt_qualifier_t`. Similar to the ptch qualifier, which uses a raw value and not an enum.
AcInLifWideData	Qualifies on the Ac InLIF wide (generic) data.

Table 43: List of iPMF2 Stage Context Selection Qualifiers

bcmFieldQualify…	Description
CascadedKeyValue	Value of key cascaded from prior group in cascade
CompareKeysResult0	Compare key result 0
CompareKeysResult1	Compare key result 1

Table 44: List of iPMF3 Stage Context Selection Qualifiers

bcmFieldQualify…	Description
ForwardingType	Qualify upon Forwarding Layer Type. Possible layer types are bcmFieldLayerTypeXXX.
VPortRangeCheck	Virtual Port Range Check Results
ContextId	Qualifies on the Context Id
ForwardingLayerQualifier	Use this qualifier together with `input_arg` to indicate which layer number relative to forwarding layer to qualify upon. It qualifies Layer Qualifiers (look for layer record qualifiers in UM).
RpfEcmpMode	RPF ECMP mode

Table 44: List of iPMF3 Stage Context Selection Qualifiers (Continued)

bcmFieldQualify…	Description
PortClassPacketProcessing	PP port profile
PortClassTrafficManagement	TM port profile
ForwardingTypeRaw	The same qualifier as bcmFieldQualifyForwardingType but does not have a mapping function. In other words, the input expects a raw value (HW value).

Table 45: List of Egress-PMF Stage Context Selection Qualifiers

bcmFieldQualify…	Description
SrcClassField	PPH Value 2
DstClassField	PPH Value 1
ForwardingType	Use this qualifier together with `input_arg` to indicate which layer number relative to forwarding layer to qualify upon. Possible layer types are bcmFieldLayerTypeXXX.
VlanFormat	Incoming vlan structure format (untagged/single/double/priority-tagged). For possible values, see bcm_port_tag_struct_type_t.
FheiSize	DNX FHEI header size in bytes
PphPresent	Qualifies on PPH present field in ITMH
UDHBase0	User-defined Header 0 encoded type and size – Value 1: type PMF size 32b
UDHBase1	User-defined Header 1 encoded type and size – Value 1: type PMF size 32b
UDHBase2	User-defined Header 2 encoded type and size – Value 1: type PMF size 32b
UDHBase3	User-defined Header 3 encoded type and size – Value 1: type PMF size 32b
ContextId	Qualifies on the Context Id (see bcmFieldForwardContextXXX)
ForwardingLayerQualifier	Use this qualifier together with `input_arg` to indicate which layer number relative to forwarding layer to qualify upon. It qualifies Layer Qualifiers (look for layer record qualifiers in UM).
PortClassPacketProcessing	PP port profile
InVportClass0	InLIF Profile 0, by Gport
VlanFormatRaw	Qualifies on the incoming VLAN structure format (untagged, single-tagged, double-tagged, or priority-tagged). The input expects a raw value.
LearnExtensionPresent	Qualifies on whether a PPH learn extension is present
ForwardingTypeRaw	The same qualifier as bcmFieldQualifyForwardingType but without the mapping function. That is, the input expects a raw value (HW value).
ContextIdRaw	Qualifies on the forwarding context value. The input expects a raw value. Relevant for egress stage only.
FtmhAsePresent	Qualify packet on FTMH ASE present field.
FtmhAseType	Qualify packet on FTMH ASE type value.

11.33.2 Qualifiers List

To see latest version support, run the diagnostic command:

```
field qualifier Predefined bcm stage=iPMF1/2/3/ePMF class=meta/layer/header
```

11.33.2.1 Header Qualifiers List

Table 46: Header Qualifiers List

bcmFieldQualify…	Description
SrcIp6	IPv6 Source address
DstIp6	IPv6 destination address
SrcIp6High	Source IPv6 Address (High/Upper 64 bits, 64 MSB)
DstIp6High	Destination IPv6 Address (High/Upper 64 bits, 64 MSB)
SrcIp6Low	Source IPv6 Address (Low/Lower 64 bits, 64 LSB)
DstIp6Low	Destination IPv6 Address (Low/Lower 64 bits, 64 LSB)
SrcMac	Source MAC address
DstMac	Destination MAC address
SrcIp	Packet Source IP address
DstIp	Packet destination IP address
Ip6FlowLabel	IPv6 Flow Label
L4SrcPort	L4 source port
L4DstPort	L4 destination port
Ip6NextHeader	Next protocol field in IPv6 Header
Ip6TrafficClass	Traffic class field in IPv6 Header
Ip6HopLimit	Hop Count field in IPv6 Header
TcpControl	TCP control Flags
IpFlags	IP Flags Field
ExtensionHeaderType	Qualifies on Next Header Field in First Extension Header
ArpSenderIp4	Sender IPv4 field of ARP header
ArpTargetIp4	Target IPv4 field of ARP header
ArpOpcode	Opcode field of ARP header
Ip4Protocol	Qualify Upon IPv4 Protocol
Ip4Tos	Qualify Upon IPv4 TOS
Ip4Ttl	Qualify Upon IPv4 TTL
EtherTypeUntagged	802.1 Ethernet Type, only for untagged frames
Tpid	VLAN TPID, offset of the value is inside the VLAN tag. Need to add the offset where the tag start in the Layer (inner/outer).
VlanId	VLAN ID, offset of the value is inside the VLAN tag. Need to add the offset where the tag start in the Layer (inner/outer).
VlanPri	VLAN P-bits , offset of the value is inside the VLAN tag. Need to add the offset where the tag start in the Layer (inner/outer).
VlanCfi	VLAN CFI , offset of the value is inside the VLAN tag. Need to add the offset where the tag start in the Layer (inner/outer).
VlanPriCfi	VLAN P-bits and CFI , offset of the value is inside the VLAN tag. Need to add the offset where the tag start in the Layer (inner/outer).
MplsLabel	MPLS Label
MplsLabelId	ID field of MPLS Label
MplsLabelTtl	TTL field of MPLS Label
MplsLabelBos	BOS field of MPLS Label
MplsLabelExp	TTL field of MPLS Label

11.33.2.2 Layer Record Qualifiers List

Table 47: Layer Record Qualifiers List

bcmFieldQualify…	Description
IpFrag	IP fragments
LayerRecordType	Qualifies on the Layer Record Type. For possible values look at bcm_field_layer_type_t
LayerRecordOffset	Qualifies on the Layer Record Offset
LayerRecordQualifier	Qualifies on the Layer Record qualifier
EthernetBroadcast	Qualify on Ethernet packets for which the destination MAC address is broadcast (MAC-DA[47:0] is all 1)
EthernetFirstTpidExist	Qualify on Ethernet packets for which their first VLAN TPID exists
EthernetFirstTpidIndex	Qualify on Ethernet packets first VLAN TPID Index
EthernetSecondTpidExist	Qualify on Ethernet packets for which their second VLAN TPID exists
EthernetSecondTpidIndex	Qualify on Ethernet packets' second VLAN TPID Index
EthernetThirdTpidExist	Qualify on Ethernet packets for which their third VLAN TPID exists
EthernetThirdTpidIndex	Qualify on Ethernet packets' third VLAN TPID Index
L2Format	Qualify on the type of the frame (see bcmFieldL2FormatXXX)
IpHasOptions	Qualify on IPv4 packets where the IP Header include options
IpFirstFrag	Qualify on IPv4 packets where the IP Header is fragmented and this is the first fragment
IpTunnelType	Qualify on IPv4 TunnelType (see bcmFieldTunnelXXX)
Ip6MulticastCompatible	Qualify on IPv6 packets for which their DIP is Multicast (8 MSBs are 0xFF)
Ip6FirstAdditionalHeaderExist	Qualify on IPv6 packets' first additional header exists
Ip6FirstAdditionalHeader	Qualify on IPv6 packets' first additional header (see bcmFieldIp6AdditionalHeaderXXX)
Ip6SecondAdditionalHeaderExist	Qualify on IPv6 packets' second additional header exists
Ip6SecondAdditionalHeader	Qualify on IPv6 packets' second additional header (see bcmFieldIp6AdditionalHeaderXXX)
ItmhPphType	Qualify on Ingress TM Header PPH type
LayerRecordTypeRaw	Qualifies on the Layer Record Type. For possible values look at bcm_field_layer_type_t; input expects raw value
Ip4DstMulticast	Qualify on IPv4 packets for which their DIP is multicast
IpTunnelTypeRaw	Raw Qualify on IPv4 TunnelType

11.33.2.3 Metadata Qualifiers List

Table 48: List of iPMF-1,2 Qualifiers

bcmFieldQualify…	Description
InPort	Ingress Port, value should be encoded as LOCAL GPORT Type. Includes core ID, and can be used by entries for all cores
L4PortRangeCheck	Qualify on L4 Ops (L4 source and dest ports) range. The qualifier is a bitmap and not specific range id, it means if range 2 was configured and the qualifier is hit, the entry value is expected to be 0x100
SrcPort	Qualify on source port, value should be encoded as SYS_PORT GPORT Type
DstPort	Qualify on destination port, all ingress stages expect raw value, egress expects GPORT encoded as SYS_PORT
PacketRes	When TRUE packet lookup result was Unknown Destination
InterfaceClassL2	VSI Profile
Vrf	VRF Id

Table 48: List of iPMF-1,2 Qualifiers (Continued)

bcmFieldQualify…	Description
VlanFormat	Incoming vlan structure format (untagged/single/double/priority-tagged). For possible values, see bcm_port_tag_struct_type_t.
DstMulticastGroup	Multicast Group id, encoded as GPORT
SrcModPortGport	Qualify on Source module/port pair, value should be encoded as SYS_PORT GPORT Type
Color	Qualify based on packet color, value should be one of BCM_FIELD_COLOR_XXX
IntPriority	Internal priority (TC)
Vpn	VSI Id
MplsForwardingLabelAction	MPLS forwarding label action
L2Learn	L2 learn enable
EcnValue	Qualify on metadata ECN or congestion information (including CNI)
RxTrapCode	Qualify on RX Trap Code
Ptch	Opaque attribute field of the Injected packets. Part of the PTCH_2 header. The input expects a raw value.
RxTrapData	Qualify on RX Trap qualifier
TrillEgressRbridge	Egress RBridge Nickname
ISid	I-SID (MAC-in-MAC lookup-id)
DstRpfGport	RPF destination (gport) for the RPF checkRpfRouteValid
TrunkHashResult	Trunk Hash Result (that is, the Load-balancing Key)
PacketLengthRangeCheck	Packet length range. The range ID on which the packet matched. If the packet size > 144B, it is treated as 144B.
LearnSrcMac	Qualifies on learn source MAC
LearnVlan	Qualifies on learn VLAN
PacketSize	Packet size in Bytes
TranslatedOuterVlanId	Translated Outer VLAN ID
TranslatedOuterVlanPri	Translated Outer VLAN priority
TranslatedOuterVlanCfi	Translated Outer VLAN CFI
TranslatedInnerVlanId	Translated inner VLAN Id
TranslatedInnerVlanPri	Translated inner VLAN priority
TranslatedInnerVlanCfi	Translated inner VLAN CFI
RxTrapCodeForSnoop	Qualify on Snoop Code
IncomingIpIfClass	RIF profile
RxTrapStrength	Qualify on RX Trap Strength
OamUpMep	Indicates if the OAM packet is vUP-MEP (sent to a destination in the network, as opposed to a specific port)
OamSubtype	In OAM the packet type is specified in the OAM header and mapped to a subtype in the hardware
OamHeaderOffset	Indicates the offset of the OAM header relative to the start of packet (as opposed to start of header-offset)
OamStampOffset	Indicates the offset to the position, in the OAM header, where the ToD or counter value should be stamped relative to the start of packet
OamMepId	If MEP is handled in OAMP, then the OAM-ID is the MEP-ID (equivalent to the index used to access the MEP DB)
OamMeterDisable	Attribute that is passed to the PMF and can also be configured by the user per MEP
VlanAction	VLAN translation action ID
GeneratedTtl	Qualify on Packet's generate Time-To-Live

Table 48: List of iPMF-1,2 Qualifiers (Continued)

bcmFieldQualify…	Description
IpMulticastCompatible	Packet is compatible for multicast
PacketIsIEEE1588	Indicating whether the packet is 1588
IEEE1588Encapsulation	IEEE-1588 Encapsulation according to bcm_field_IEEE1588Encap_t
IEEE1588CompensateTimeStamp	IEEE-1588 update time stamp
IEEE1588Command	Command used by egress pipeline, indicating if CF(correction field) needs to be update
IEEE1588HeaderOffset	This field indicates the offset in bytes of the IEEE-1588 header
KeyGenVar	Match on configured key Gen Variable (values that was configured for context)
NetworkQos	Qualify Upon Network Qos
EcmpLoadBalanceKey0	ECMP Load Balance Key 0
EcmpLoadBalanceKey1	ECMP Load Balance Key 1
EcmpLoadBalanceKey2	ECMP Load Balance Key 2
ForwardingLayerIndex	Qualify on the Forwarding Layer Index
AcInLifWideData	Qualifies on the Ac InLIF wide(generic) data
NativeAcInLifWideData	Qualifies on the native Ac InLIF wide(generic) data
TracePacket	Qualifies upon Packets that Trace was set for them
InInterface	Ingress Logical Interface (LIF/RIF), value should be encoded as L3 interface
OutInterface	Egress Logical Interface (LIF/RIF), should be encoded as L3 interface
PortClassPacketProcessing	PP port profile
PortClassTrafficManagement	TM port profile
InVPort0	In LIF 0, value should be GPORT with subtype LIF
InVPort1	In LIF 1, value should be GPORT with subtype LIF
OutVPort0	OutLIF 0, value should be GPORT with subtype LIF
OutVPort1	OutLIF 1, value should be GPORT with subtype LIF
BierStringOffset	Qualifies upon Bier string offset
BierStringSize	Qualifies upon Bier string size
PacketIsBier	Qualifies upon Bier packets
PortClassPacketProcessingGeneralData	PP port general data
ForwardingAdditionalInfo	Qualifies on the Forwarding additional info
InVportClass0	InLIF Profile 0, by Gport
InVportClass1	InLIF Profile 1, by Gport
StatId0	Qualifies on statistics ID 0
StatId1	Qualifies on statistics ID 1
StatId2	Qualifies on statistics ID 2
StatId3	Qualifies on statistics ID 3
StatId4	Qualifies on statistics ID 4
StatId5	Qualifies on statistics ID 5
StatId6	Qualifies on statistics ID 6
StatId7	Qualifies on statistics ID 7
StatId8	Qualifies on statistics ID 8
StatId9	Qualifies on statistics ID 9
StatProfile0	Qualifies on statistics profile 0

Table 48: List of iPMF-1,2 Qualifiers (Continued)

bcmFieldQualify…	Description
StatProfile1	Qualifies on statistics profile 1
StatProfile2	Qualifies on statistics profile 2
StatProfile3	Qualifies on statistics profile 3
StatProfile4	Qualifies on statistics profile 4
StatProfile5	Qualifies on statistics profile 5
StatProfile6	Qualifies on statistics profile 6
StatProfile7	Qualifies on statistics profile 7
StatProfile8	Qualifies on statistics profile 8
StatProfile9	Qualifies on statistics profile 9
StatMetaData	Qualifies on Statistics metadata
InVPort0Raw	Qualifies on In LIF 0 input expects raw value
InVPort1Raw	Qualifies on In LIF 1 input expects raw value
OutVPort0Raw	Qualifies on OutLIF 0 input expects raw value
OutVPort1Raw	Qualifies on OutLIF 1 input expects raw value
MirrorCode	Matches on Mirror Code
LearnVsi	Qualifies on learn VSI
LearnData	Qualifies on data field in the Learn Info
LearnStationMove	Qualifies on learn Station move
LearnMatch	Qualifies on learn Match
LearnFound	Qualifies on learn Found
LearnExpectedWon	Qualifies on learn Expected Won
VrfValue	Qualifies on the VRF part of forwarding data, without checking that the type is VRF
InPortWithoutCore	Ingress Port, value should be encoded as LOCAL GPORT Type. Does not include the core ID, caution should be taken when not adding entry by core
SaLookupAcceptedStrength	Qualifies on Accepted strength LSB returned from SA Lookup result. Qualifier is only relevant for bridged packets
SrcPortRaw	Qualifies on source port, input expects raw value
InPortWithoutCoreRaw	Qualifies on Ingress Port. Does not include the core ID, caution should be taken when not adding entry by core, input expects raw value
VlanFormatRaw	Qualifies on the incoming VLAN structure format (untagged, single tagged, double-tagged, or priority-tagged). The input expects a raw value
InPortRaw	Qualifies on Ingress Port. Includes core ID, and can be used by entries for all cores, input expects raw value
ColorRaw	Qualifies on packet color, input expects raw value
DstRpfValid	Checks if the RPF Destination is valid
RpfOutVPort	Qualifies on the RPF OUT LIF. Value should be GPORT with subtype LIF
RpfOutInterface	Qualifies on the RPF OUT LIF. Value should be L3 interface
RpfOutVPortRaw	Qualifies on the RPF OUT LIF
RpfRouteValid	Qualifies on the RPF Route valid
VipValid	Qualifies on SLLB Virtual Wire VW_VIP_VALID
VipId	Qualifies on SLLB Virtual Wire VW_VIP_ID
VIPMemberReference	Qualifies on SLLB Virtual Wire VW_MEMBER_REFERENCE
PccHit	Qualifies on SLLB Virtual Wire VW_PCC_HIT

Table 48: List of iPMF-1,2 Qualifiers (Continued)

bcmFieldQualify…	Description
PrtQualifier	Qualifies on the prt_qualifier signal, which may be set by the injected packet (Opaque attribute field part of the PTCH_2 header). The input expects an enum value `bcm_field_prt_qualifier_t`. Similar to the ptch qualifier, which uses a raw value and not an enum.

Table 49: List of iPMF-3 Qualifiers

bcmFieldQualify…	Description
InPort	Ingress Port, value should be encoded as LOCAL GPORT Type. Includes core ID, and can be used by entries for all cores
SrcPort	Qualify on source port, value should be encoded as SYS_PORT GPORT Type
DstPort	Qualify on destination port, all ingress stages expect raw value, egress expects GPORT encoded as SYS_PORT
InterfaceClassL2	VSI Profile
Vrf	VRF Id
SrcModPortGport	Qualify on Source module/port pair, value should be encoded as SYS_PORT GPORT Type
Color	Qualify based on packet color, value should be one of BCM_FIELD_COLOR_XXX
IntPriority	Internal priority (TC)
Vpn	VSI Id
FlowId	Qualify on destination Flow-Id
MplsForwardingLabelAction	MPLS forwarding label action
L2Learn	L2 learn enable
EcnValue	Qualify on metadata ECN or congestion information (including CNI)
RxTrapCode	Qualify on RX Trap Code
RxTrapData	Qualify on RX Trap qualifier
TrillEgressRbridge	Egress RBridge Nickname
ISid	I-SID (MAC-in-MAC lookup-id)
DstRpfGport	RPF destination (gport) for the RPF check
TrunkHashResult	Trunk Hash Result (i.e., the Load-balancing Key)
LearnSrcMac	Qualifies on learn source MAC
LearnVlan	Qualifies on learn VLAN
TranslatedOuterVlanId	Translated Outer VLAN ID
TranslatedOuterVlanPri	Translated Outer VLAN priority
TranslatedOuterVlanCfi	Translated Outer VLAN CFI
TranslatedInnerVlanId	Translated inner VLAN Id
TranslatedInnerVlanPri	Translated inner VLAN priority
TranslatedInnerVlanCfi	Translated inner VLAN CFI
DstGport	Qualifies packet according to destination encoded as trap code
RxTrapCodeForSnoop	Qualify on Snoop Code
IncomingIpIfClass	RIF profile
StackingRoute	Stacking Route History bitmap
RxTrapStrength	Qualify on RX Trap Strength

Table 49: List of iPMF-3 Qualifiers (Continued)

bcmFieldQualify…	Description
OamUpMep	Indicates if the OAM packet is vUP-MEP (sent to a destination in the network, as opposed to a specific port)
OamSubtype	In OAM the packet type is specified in the OAM header and mapped to a subtype in the hardware
OamHeaderOffset	Indicates the offset of the OAM header relative to the start of packet (as opposed to start of header-offset)
OamStampOffset	Indicates the offset to the position, in the OAM header, where the ToD or counter value should be stamped relative to the start of packet
OamMepId	If MEP is handled in OAMP, then the OAM-ID is the MEP-ID (equivalent to the index used to access the MEP DB)
VlanAction	VLAN translation action ID
GeneratedTtl	Qualify on Packet's generate Time-To-Live
PacketIsIEEE1588	Indicating whether the packet is 1588
IEEE1588Encapsulation	IEEE-1588 Encapsulation according to bcm_field_IEEE1588Encap_t
IEEE1588CompensateTimeStamp	IEEE-1588 update time stamp
IEEE1588Command	Command used by egress pipeline, indicating if CF(correction field) needs to be update
IEEE1588HeaderOffset	This field indicates the offset in bytes of the IEEE-1588 header
KeyGenVar	Match on configured key Gen Variable (values that was configured for context)
Container	This qualifier will be used as container in IPMF3, to receive the action buffer, when performing cascading between IPMF1/2 and IPMF3
UDHBase0	User-defined Header 0 encoded type and size – Value 1: type PMF size 32b
UDHBase1	User-defined Header 1 encoded type and size – Value 1: type PMF size 32b
UDHBase2	User-defined Header 2 encoded type and size – Value 1: type PMF size 32b
UDHBase3	User-defined Header 3 encoded type and size – Value 1: type PMF size 32b
NetworkQos	Qualify Upon Network Qos
ForwardingLayerIndex	Qualify on the forwarding layer index
IPTProfile	Qualifies upon IPT Profile (End Of Packet Editing)
UDHData0	User-defined Header 0 payload MS bits of UDH
UDHData1	User-defined Header 1 payload
UDHData2	User-defined Header 2 payload
UDHData3	User-defined Header 3 payload LS bits of UDH
RxSnoopStrength	Snoop Strength
StatSamplingCode	Statistical Sampling Code value
StatSamplingQualifier	Statistical Sampling qualifier value
RpfEcmpMode	RPF ECMP mode
StatOamLM	OAM LM counter value, the index of the values are bcmFieldStatOamLmIndexXXX
InInterface	Ingress Logical Interface (LIF/RIF), value should be encoded as L3 interface
OutInterface	Egress Logical Interface (LIF/RIF), should be encoded as L3 interface
PortClassPacketProcessing	PP port profile
PortClassTrafficManagement	TM port profile
InVPort0	In LIF 0, value should be GPORT with subtype LIF
InVPort1	In LIF 1, value should be GPORT with subtype LIF
OutVPort0	OutLIF 0, value should be GPORT with subtype LIF
OutVPort1	OutLIF 1, value should be GPORT with subtype LIF

Table 49: List of iPMF-3 Qualifiers (Continued)

bcmFieldQualify…	Description
OutVPort2	OutLIF 2, value should be GPORT with subtype LIF
OutVPort3	OutLIF 3, value should be GPORT with subtype LIF
BierStringOffset	Qualifies upon Bier string offset
BierStringSize	Qualifies upon Bier string size
PacketIsBier	Qualifies upon Bier packets
PortClassPacketProcessingGeneralData	PP port general data
ForwardingAdditionalInfo	Qualifies on the Forwarding additional info
InVportClass0	InLIF Profile 0, by Gport
InVportClass1	InLIF Profile 1, by Gport
StatMetaData	Qualifies on Statistics metadata
InVPort0Raw	Qualifies on In LIF 0 input expects raw value
InVPort1Raw	Qualifies on In LIF 1 input expects raw value
OutVPort0Raw	Qualifies on OutLIF 0 input expects raw value
OutVPort1Raw	Qualifies on OutLIF 1 input expects raw value
OutVPort2Raw	Qualifies on OutLIF 2 input expects raw value
OutVPort3Raw	Qualifies on OutLIF 3 input expects raw value
MirrorCode	Matches on Mirror Code
MirrorData	Matches on Mirror Data (qualifier)
LearnVsi	Qualifies on learn VSI
LearnData	Qualifies on data field in the Learn Info
LearnStationMove	Qualifies on learn Station move
LearnMatch	Qualifies on learn Match
LearnFound	Qualifies on learn Found
LearnExpectedWon	Qualifies on learn Expected Won
VrfValue	Qualifies on the VRF part of forwarding data, without checking that the type is VRF
InPortWithoutCore	Ingress Port, value should be encoded as LOCAL GPORT Type. Does not include the core ID, caution should be taken when not adding entry by core
SrcPortRaw	Qualifies on source port, input expects raw value
InPortWithoutCoreRaw	Qualifies on Ingress Port. Does not include the core ID, caution should be taken when not adding entry by core, input expects raw value
InPortRaw	Qualifies on Ingress Port. Includes core ID, and can be used by entries for all cores, input expects raw value
ColorRaw	Qualifies on packet color, input expects raw value
DstRpfValid	Checks if the RPF Destination is valid
EcmpGroup	Qualifies on the ECMP group
StatOamLMRaw	OAM LM counter raw value

Table 50: List of Egress-PMF Qualifiers

bcmFieldQualify…	Description
L4PortRangeCheck	Qualify on L4 Ops (L4 source and dest ports) range. The qualifier is a bitmap and not specific range id, it means if range 2 was configured and the qualifier is hit, the entry value is expected to be 0x100
SrcPort	Qualify on source port, value should be encoded as SYS_PORT GPORT Type

Table 50: List of Egress-PMF Qualifiers (Continued)

bcmFieldQualify…	Description
DstPort	Qualify on destination port, all ingress stages expect raw value, egress expects GPORT encoded as SYS_PORT
SrcClassField	PPH Value 2
DstClassField	PPH Value 1
InterfaceClassL2	VSI Profile
Vrf	VRF Id
OutPort	Egress port, value should be encoded as LOCAL GPORT Type. Does not include the core ID, caution should be taken when not adding entry by core
DstL3Egress	L3 Egress Interface, Raw value of SYS_PORT
DstMulticastGroup	Multicast Group id, encoded as GPORT
SrcModPortGport	Qualify on Source module/port pair, value should be encoded as SYS_PORT GPORT Type
Color	Qualify based on packet color, value should be one of BCM_FIELD_COLOR_XXX
IntPriority	Internal priority (TC)
Vpn	VSI Id
RepCopy	qualify on Copy type: 0:Forwarded / 1:Snooped / 2:Mirrored / 3:Statistical Sampling, packet
EcnValue	Qualify on metadata ECN or congestion information (including CNI)
RxTrapCode	Qualify on RX Trap Code
TrunkHashResult	Trunk Hash Result (i.e., the Load-balancing Key)
LearnSrcMac	Qualifies on learn source MAC
LearnVlan	Qualifies on learn VLAN
Fhei	DNX FHEI header field
FheiSize	DNX FHEI header size in bytes
StackingRoute	Stacking Route History bitmap
RxTrapStrength	Qualify on RX Trap Strength
OamTsSystemHeader	FTMH Application specific extension
DstSysPortExt	Qualifies upon TM Destination extension header
GeneratedTtl	Qualify on Packet's generate Time-To-Live
PphPresent	Qualifies on PPH present field in ITMH
PacketProcessingInVportClass	Qualifies on InLIF profile
UDHBase0	User-defined Header 0 encoded type and size – Value 1: type PMF size 32b
UDHBase1	User-defined Header 1 encoded type and size – Value 1: type PMF size 32b
UDHBase2	User-defined Header 2 encoded type and size – Value 1: type PMF size 32b
UDHBase3	User-defined Header 3 encoded type and size – Value 1: type PMF size 32b
NetworkQos	Qualify Upon Network Qos
AceEntryId	Qualify Upon Ace Entry ID
NetworkLoadBalanceKey	Network Load Balance Key
IPTProfile	Qualifies upon IPT Profile (End Of Packet Editing)
UDHData0	User-defined Header 0 payload MS bits of UDH
UDHData1	User-defined Header 1 payload
UDHData2	User-defined Header 2 payload
UDHData3	User-defined Header 3 payload LS bits of UDH
RxSnoopStrength	Snoop Strength
RxSnoopCode	qualify upon Snoop profile

Table 50: List of Egress-PMF Qualifiers (Continued)

bcmFieldQualify…	Description
InInterface	Ingress Logical Interface (LIF/RIF), value should be encoded as L3 interface
OutInterface	Egress Logical Interface (LIF/RIF), should be encoded as L3 interface
OutVportClass	OutLIF Profile, by Gport
PortClassPacketProcessing	PP port profile
PortClassTrafficManagement	TM port profile
OutPortTrafficManagement	Out TM-port
InVPort0	In LIF 0, value should be GPORT with subtype LIF
OutVPort0	OutLIF 0, value should be GPORT with subtype LIF
OutVPort1	OutLIF 1, value should be GPORT with subtype LIF
OutVPort2	OutLIF 2, value should be GPORT with subtype LIF
OutVPort3	OutLIF 3, value should be GPORT with subtype LIF
ForwardingAdditionalInfo	Qualifies on the Forwarding additional info
InVportClass0	InLIF Profile 0, by Gport
InVPort0Raw	Qualifies on In LIF 0 input expects raw value
OutVPort0Raw	Qualifies on OutLIF 0 input expects raw value
OutVPort1Raw	Qualifies on OutLIF 1 input expects raw value
OutVPort2Raw	Qualifies on OutLIF 2 input expects raw value
OutVPort3Raw	Qualifies on OutLIF 3 input expects raw value
LearnVsi	Qualifies on learn VSI
LearnData	Qualifies on the data field in the Learn Info
VrfValue	Qualifies on the VRF part of forwarding data, without checking that the type is VRF
DstSysPortExtPresent	Qualifies upon if TM Destination extension header is present
SrcPortRaw	Qualifies on source port, input expects raw value
ColorRaw	Qualifies on packet color, input expects raw value
DstPortRaw	Qualifies on destination port, input expects raw value
OutPortRaw	Qualifies on Egress port. Does not include the core ID, caution should be taken when not adding entry by core, input expects raw value
McastPkt	All ingress replicas will have this value set to 1
LearnExtensionPresent	Qualifies on whether PPH learn extension is present
FtmhAsePresent	Qualify packet on FTMH ASE present field

NOTE: Qualifiers sizes can be obtained by using the following shell command:

```
field map qualifier bcm <stage=ipmf1/2/3..epmf>
```

11.33.3 Actions List

To see latest version support, run the following diagnostic command:

```
field action Predefined stage=iPMF1/2/3/ePMF
```

Table 51: iPMF-1,2 Actions

bcmFieldAction…	Description
PrioIntNew	TC update
EcnNew	Change metadata ECN or congestion information (including CNI)

Table 51: iPMF-1,2 Actions (Continued)

bcmFieldAction…	Description
Redirect	Redirect packet to to destination
RedirectMcast	Redirect to Multicast Group Id
MirrorIngress	Set mirror, value should be encoded as GPORT Type Mirror for mirror command
L3Switch	Sets packet destination, input expects L3 interface
DropPrecedence	Set Packet's Drop Precedence, the values are BCM_FIELD_COLOR_XXX
VrfSet	Set VRF. value[0]: fwd_domain_id_type. value[1]: fwd_domain_id_value. value[1]: fwd_domain_profile. In iPMF3, fwd_domain_profile isn't updated
InnerVlanNew	Replace inner VLAN Id
OuterVlanNew	Replace outer VLAN Id
OuterVlanPrioNew	Replace outer VLAN tag priority
VSQ	Assign matching packets to specified VSQ
Trap	Set Trap code, use TRAP GPORT type value[0]=TRAP_GPORT value[1]=Trap Qualifier. at egress stage refers only to the TRAP ID (strength is not relevant, and can be changed using bcmFieldActionTrapStrength action)
UsePolicerResult	Specify/override where policer result will be used for matched packets
Snoop	Snoop matched packets, input can be either a snoop command: value[0] = Mirror Gport, value[1]=snoop qualifier, value[2]=snoop strength OR: snoop code: value[0] = Trap Gport, value[1]=snoop qualifier, at egress stage refers only to the snoop ID (strength is not relevant, and can be changed using bcmFieldActionSnoopStrength action)
TtlSet	New TTL
DstRpfGportNew	Set the RPF Destination
SrcGportNew	Set the Source-Port
SystemHeaderSet	Modify EEI
VSwitchNew	New VSI. value[0]: fwd_domain_id_type. value[1]: fwd_domain_id_value. value[1]: fwd_domain_profile. In iPMF3, fwd_domain_profile isn't updated
VlanActionSetNew	Modify the VLAN Action Set Id
IngressGportSet	Set a new ingress Global Lif encoded, Values should be Encoded as GPORT
StackingRouteNew	Replace the value of the stacking route
PphPresentSet	If set, a packet processing header is present
Oam	Changes 4 OAM signals. value[0]: OAM-Stamp-Offset. value[1]: OAM-offset. value[2]: OAM-Sub-Type. value[3]: OAM-Up-Mep
PacketTraceEnable	Enable Trace packet
IEEE1588	Setting various parameters for 1588 frames. Bits(0:7): header_offset. Bit(8): encapsulation. Bits(9:10) command. Bit(11): compensate_time_stamp. Bit(12): packet_is_ieee1588
Forward	Forward Destination Raw value, setting to 1's will drop the packet
PphSnoopCode	Set the PPH Snoop code value (Reserved Bits)
IngressTimeStampInsert	Insert IPIPE time stamp
AdmitProfile	Admit Profile
LatencyFlowId	Change the latency flow-Id. value[0] valid. value[1]: latency_flow_id. value[2]: latency_flow_profile
VisibilityEnable	Enables visibility for current packet
InVportClass0	Updates the profile of an in LIF encoded as Gport
InVportClass1	Updates the profile of an in LIF encoded as Gport
NetworkQos	Updates Network Qos value
Void	Void action, used for cascading, saving buffer in the action payload (for FEM use)

Table 51: iPMF-1,2 Actions (Continued)

bcmFieldAction…	Description
Container	This action will be used as container in IPMF2, to parse the action buffer, when performing cascading between IPMF2 and IPMF3
EgressForwardingIndex	Indicate for egress parser which layer is the Forwarding layer
UDHData0	Set the Payload for User-defined Header 0
UDHData1	Set the Payload for User-defined Header 1
UDHData2	Set the Payload for User-defined Header 2
UDHData3	Set the Payload for User-defined Header 3
UDHBase0	Set the Base for User-defined Header 0
UDHBase1	Set the Base for User-defined Header 1
UDHBase2	Set the Base for User-defined Header 2
UDHBase3	Set the Base for User-defined Header 3
IPTProfile	IPT Profile action (used for instrumentation)
StatId0	Updates statistics ID 0
StatId1	Updates statistics ID 1
StatId2	Updates statistics ID 2
StatId3	Updates statistics ID 3
StatId4	Updates statistics ID 4
StatId5	Updates statistics ID 5
StatId6	Updates statistics ID 6
StatId7	Updates statistics ID 7
StatId8	Updates statistics ID 8
StatId9	Updates statistics ID 9
StatProfile0	Updates statistics profile 0. value[0]: is_meter. value[1]: is_lm. value[2]: type. value[3]: valid
StatProfile1	Updates statistics profile 1. value[0]: is_meter. value[1]: is_lm. value[2]: type. value[3]: valid
StatProfile2	Updates statistics profile 2. value[0]: is_meter. value[1]: is_lm. value[2]: type. value[3]: valid
StatProfile3	Updates statistics profile 3. value[0]: is_meter. value[1]: is_lm. value[2]: type. value[3]: valid
StatProfile4	Updates statistics profile 4. value[0]: is_meter. value[1]: is_lm. value[2]: type. value[3]: valid
StatProfile5	Updates statistics profile 5. value[0]: is_meter. value[1]: is_lm. value[2]: type. value[3]: valid
StatProfile6	Updates statistics profile 6. value[0]: is_meter. value[1]: is_lm. value[2]: type. value[3]: valid
StatProfile7	Updates statistics profile 7. value[0]: is_meter. value[1]: is_lm. value[2]: type. value[3]: valid
StatProfile8	Updates statistics profile 8. value[0]: is_meter. value[1]: is_lm. value[2]: type. value[3]: valid
StatProfile9	Updates statistics profile 9. value[0]: is_meter. value[1]: is_lm. value[2]: type. value[3]: valid
InvalidNext	Invalid next Action macro Id
IngressDoNotLearn	Disable Ingress learning
EgressDoNotLearn	Disable Egress learning
VisibilityClear	Disables visibility for current packet
StatMetaData	Set statistics metaData
InVport0	Change the InLIF 0, value should be encoded as a GPORT with LIF subtype
InVport1	Change the InLIF 1, value should be encoded as a GPORT with LIF subtype
OutVport0	Change OutLIF 0, value should be encoded as a GPORT with LIF subtype
OutVport1	Change OutLIF 1, value should be encoded as a GPORT with LIF subtype
OutVport2	Change OutLIF 2, value should be encoded as a GPORT with LIF subtype
OutVport3	Change OutLIF 3, value should be encoded as a GPORT with LIF subtype

Table 51: iPMF-1,2 Actions (Continued)

bcmFieldAction…	Description
ForwardingTypeMeterMap	Maps L2 forwarding type to one of the general meters
StatSampling	Update statistical sampling value, value encoded as Mirror Gport
StatOamLM	Update the counter of OAM LM, the index of the values are bcmFieldStatOamLmIndexXXX
ForwardingLayerIndex	Update forwarding layer index
InInterface0	Change the InLIF 0, value should be encoded as an interface
InInterface1	Change the InLIF 1, value should be encoded as an interface
OutInterface0	Change OutLIF 0, value should be encoded as type interface
OutInterface1	Change OutLIF 1, value should be encoded as type interface
OutInterface2	Change OutLIF 2, value should be encoded as type interface
OutInterface3	Change OutLIF 3, value should be encoded as type interface
BierStringOffset	Updates Bier string offset
BierStringSize	Updates Bier string size
PacketIsBier	Updates is Bier packet
Eventor	Enable push data to eventor
ForwardingAdditionalInfo	Updates forwarding additional information
SmallExemLearn	Updates the Small Exact Match learn info (IPMF2 only)
InVport0Raw	Change the InLIF 0, with raw value
InVport1Raw	Change the InLIF 1, with raw value
OutVport0Raw	Change the OutLIF 0, with raw value
OutVport1Raw	Change the OutLIF 1, with raw value
OutVport2Raw	Change the OutLIF 2, with raw value
OutVport3Raw	Change the OutLIF 3, with raw value
IngressTimeStampInsertValid	Insert ingress timestamp. Overrides the time stamp in TSH header
SnoopRaw	Snoop matched packets, input should be raw value. At egress stage refers only to the snoop ID (strength is not relevant, and can be changed using bcmFieldActionSnoopStrength action)
DropPrecedenceRaw	Set Packet's Drop Precedence, input expects raw value
FabricHeaderSetRaw	Change System Header Profile, input expects raw value
SrcGportNewRaw	Set the Source-Port, input expects raw value
LatencyFlowIdRaw	Change the latency flow-Id, input expects raw value
MirrorIngressRaw	Set mirror in ingress, input expects raw value
OamRaw	Changes 4 OAM signals (OAM-Stamp-Offset, OAM-offset, OAM-Sub-Type, OAM-Up-Mep), input expects raw value
TrapRaw	Set Trap code, input should be raw value. At egress stage refers only to the TRAP ID (strength is not relevant, and can be changed using bcmFieldActionTrapStrength action)
VrfSetRaw	Set VRF, input expects raw value
StatProfile0Raw	Updates statistics profile 0. Input expects raw value
StatProfile1Raw	Updates statistics profile 1. Input expects raw value
StatProfile2Raw	Updates statistics profile 2. Input expects raw value
StatProfile3Raw	Updates statistics profile 3. Input expects raw value
StatProfile4Raw	Updates statistics profile 4. Input expects raw value
StatProfile5Raw	Updates statistics profile 5. Input expects raw value
StatProfile6Raw	Updates statistics profile 6. Input expects raw value
StatProfile7Raw	Updates statistics profile 7. Input expects raw value

Table 51: iPMF-1,2 Actions (Continued)

bcmFieldAction…	Description
StatProfile8Raw	Updates statistics profile 8. Input expects raw value
StatProfile9Raw	Updates statistics profile 9. Input expects raw value
StatSamplingRaw	Update statistical sampling value, input expects raw value
IpMulticastCompatible	Designates when a Compatible MC procedure found a match
RpfOutVport	Updates the RPF OUT LIF. value[0]: route_valid, value[1]: default_route_found, value[2]: rpf_out_lif (encoded as GPORT with LIF subtype)
RpfOutInterface	Updates the RPF OUT LIF. value[0]: route_valid, value[1]: default_route_found, value[2]: rpf_out_lif (encoded as L3 interface)
RpfOutVportRaw	Updates the RPF OUT LIF. Input expects raw value
LearnRaw0	Updates learn info 0
LearnRaw1	Updates learn info 1
LearnRaw2	Updates learn info 2
LearnRaw3	Updates learn info 3
LearnRaw4	Updates learn info 4
DstRpfGportNewValid	Set the RPF Destination Valid
DstRpfGportNewRaw	Set the RPF Destination. Input expects raw value
StatOamLMRaw	Update the counter of OAM LM with a raw value
LearnKey0	Updates bits 31:0 of learn info key
LearnKey1	Updates bits 63:32 of learn info key
LearnKey2	Updates learn info key and app_db_index. value[0]: bits 79:64 of learn info ley, value[1] app_db_index
LearnKey2Raw	Updates learn info key and app_db_index. Key is at the LSB
LearnPayload0	Updates bits 31:0 of learn info data (payload)
LearnPayload1	Updates bits 63:32 of learn info data (payload)
LearnEntryFormatIndex	Updates entry format index in learn info
LearnOrTransplant	Updates learn or transplant bit
NetworkLoadBalanceKey	Set the network load balancing Key

Table 52: iPMF-3 Actions

bcmFieldAction…	Description
PrioIntNew	TC update
EcnNew	Change metadata ECN or congestion information (including CNI)
Redirect	Redirect packet to to destination
RedirectMcast	Redirect to Multicast Group Id
MirrorIngress	set mirror, value should be encoded as GPORT Type Mirror for mirror command
L3Switch	Sets packet destination, input expects L3 interface
DropPrecedence	Set Packet's Drop Precedence, the values are BCM_FIELD_COLOR_XXX
VrfSet	Set VRF. value[0]: fwd_domain_id_type. value[1]: fwd_domain_id_value. value[1]: fwd_domain_profile. In iPMF3, fwd_domain_profile isn't updated
InnerVlanNew	Replace inner VLAN Id
OuterVlanNew	Replace outer VLAN Id
OuterVlanPrioNew	Replace outer VLAN tag priority
VSQ	Assign matching packets to specified VSQ

Table 52: iPMF-3 Actions (Continued)

bcmFieldAction…	Description
Trap	Set Trap code, use TRAP GPORT type value[0]=TRAP_GPORT value[1]=Trap Qualifier. at egress stage refers only to the TRAP ID (strength is not relevant, and can be changed using bcmFieldActionTrapStrength action)
UsePolicerResult	Specify/override where policer result will be used for matched packets
Snoop	Snoop matched packets, input can be either a snoop command: value[0] = Mirror Gport, value[1]=snoop qualifier, value[2]=snoop strength OR: snoop code: value[0] = Trap Gport, value[1]=snoop qualifier, at egress stage refers only to the snoop ID (strength is not relevant, and can be changed using bcmFieldActionSnoopStrength action)
TtlSet	New TTL
TrunkHashKeySet	Set the Trunk Hash Key
SrcGportNew	Set the Source-Port
StartPacketStrip	Strip the start of packet. value[0]: bcm_field_start_packet_strip_t value[1]: extra bytes to remove
SystemHeaderSet	Modify EEI
VSwitchNew	New VSI. value[0]: fwd_domain_id_type. value[1]: fwd_domain_id_value. value[1]: fwd_domain_profile. In iPMF3, fwd_domain_profile isn't updated
VlanActionSetNew	Modify the VLAN Action Set Id
IngressGportSet	Set a new ingress Global Lif encoded, Values should be Encoded as GPORT
StackingRouteNew	Replace the value of the stacking route
PphPresentSet	If set, a packet processing header is present
FabricHeaderSet	Change System Header Profile, bcm_field_system_header_profile_t
Oam	Changes 4 OAM signals. value[0]: OAM-Stamp-Offset. value[1]: OAM-offset. value[2]: OAM-Sub-Type. value[3]: OAM-Up-Mep
IEEE1588	Setting various parameters for 1588 frames. Bits(0:7): header_offset. Bit(8): encapsulation. Bits(9:10) command. Bit(11): compensate_time_stamp. Bit(12): packet_is_ieee1588
Forward	Forward Destination Raw value, setting to 1's will drop the packet
PphSnoopCode	Set the PPH Snoop code value (Reserved Bits)
AdmitProfile	Admit Profile
LatencyFlowId	Change the latency flow-Id. value[0] valid. value[1]: latency_flow_id. value[2]: latency_flow_profile
VisibilityEnable	Enables visibility for current packet
SystemHeaderSizeAdjust	System Header Size adjust
InVportClass0	Updates the profile of an in LIF encoded as Gport
InVportClass1	Updates the profile of an in LIF encoded as Gport
NetworkQos	Updates Network Qos value
Void	Void action for cascading, saving buffer in the action payload (for FEM use)
ParsingStartType	Indicates the layer type that egress parser should consider. For information about the possible layer types, see Section 11.30, Common Types.
ParsingStartOffset	Indicate from which offset egress parser should start parsing (all info before that will not be parsed)
EgressForwardingIndex	Indicate for egress parser which layer is the Forwarding layer
UDHData0	Set the Payload for User-defined Header 0
UDHData1	Set the Payload for User-defined Header 1
UDHData2	Set the Payload for User-defined Header 2
UDHData3	Set the Payload for User-defined Header 3
UDHBase	Set the Base for All User-defined Header 0-3, UDHBase0 is at LSB
IPTProfile	IPT Profile action (used for instrumentation)
IPTCommand	IPT Profile command. value[0]: int_commnad. value[1]: int_profile

Table 52: iPMF-3 Actions (Continued)

bcmFieldAction…	Description
StatId0	Updates statistics ID 0
StatId1	Updates statistics ID 1
StatId2	Updates statistics ID 2
StatId3	Updates statistics ID 3
StatId4	Updates statistics ID 4
StatId5	Updates statistics ID 5
StatId6	Updates statistics ID 6
StatId7	Updates statistics ID 7
StatId8	Updates statistics ID 8
StatId9	Updates statistics ID 9
StatProfile0	Updates statistics profile 0. value[0]: is_meter. value[1]: is_lm. value[2]: type. value[3]: valid
StatProfile1	Updates statistics profile 1. value[0]: is_meter. value[1]: is_lm. value[2]: type. value[3]: valid
StatProfile2	Updates statistics profile 2. value[0]: is_meter. value[1]: is_lm. value[2]: type. value[3]: valid
StatProfile3	Updates statistics profile 3. value[0]: is_meter. value[1]: is_lm. value[2]: type. value[3]: valid
StatProfile4	Updates statistics profile 4. value[0]: is_meter. value[1]: is_lm. value[2]: type. value[3]: valid
StatProfile5	Updates statistics profile 5. value[0]: is_meter. value[1]: is_lm. value[2]: type. value[3]: valid
StatProfile6	Updates statistics profile 6. value[0]: is_meter. value[1]: is_lm. value[2]: type. value[3]: valid
StatProfile7	Updates statistics profile 7. value[0]: is_meter. value[1]: is_lm. value[2]: type. value[3]: valid
StatProfile8	Updates statistics profile 8. value[0]: is_meter. value[1]: is_lm. value[2]: type. value[3]: valid
StatProfile9	Updates statistics profile 9. value[0]: is_meter. value[1]: is_lm. value[2]: type. value[3]: valid
InvalidNext	Invalid next Action macro Id
StatMetaData	Set statistics metaData
InVport0	Change the InLIF 0, value should be encoded as a GPORT with LIF subtype
InVport1	Change the InLIF 1, value should be encoded as a GPORT with LIF subtype
OutVport0	Change OutLIF 0, value should be encoded as a GPORT with LIF subtype
OutVport1	Change OutLIF 1, value should be encoded as a GPORT with LIF subtype
OutVport2	Change OutLIF 2, value should be encoded as a GPORT with LIF subtype
OutVport3	Change OutLIF 3, value should be encoded as a GPORT with LIF subtype
ForwardingTypeMeterMap	Maps L2 forwarding type to one of the general meters
StatSampling	Update statistical sampling value, value encoded as Mirror Gport
StatOamLM	Update the counter of OAM LM, the index of the values are bcmFieldStatOamLmIndexXXX
ForwardingLayerIndex	Update forwarding layer index
InInterface0	Change the InLIF 0, value should be encoded as an interface
InInterface1	Change the InLIF 1, value should be encoded as an interface
OutInterface0	Change OutLIF 0, value should be encoded as type interface
OutInterface1	Change OutLIF 1, value should be encoded as type interface
OutInterface2	Change OutLIF 2, value should be encoded as type interface
OutInterface3	Change OutLIF 3, value should be encoded as type interface
BierStringOffset	Updates Bier string offset
BierStringSize	Updates Bier string size
PacketIsBier	Updates is Bier packet
ForwardingAdditionalInfo	Updates forwarding additional information
InVport0Raw	Change the InLIF 0, with raw value

Table 52: iPMF-3 Actions (Continued)

bcmFieldAction…	Description
InVport1Raw	Change the InLIF 1, with raw value
OutVport0Raw	Change the OutLIF 0, with raw value
OutVport1Raw	Change the OutLIF 1, with raw value
OutVport2Raw	Change the OutLIF 2, with raw value
OutVport3Raw	Change the OutLIF 3, with raw value
AppendPointerCompensation	Configures pointer to header append compensation value
IngressTimeStampInsertValid	Insert ingress timestamp. Overrides the time stamp in TSH header
SnoopRaw	Snoop matched packets, input should be raw value. At egress stage refers only to the snoop ID (strength is not relevant, and can be changed using bcmFieldActionSnoopStrength action)
DropPrecedenceRaw	Set Packet's Drop Precedence, input expects raw value
FabricHeaderSetRaw	Change System Header Profile, input expects raw value
SrcGportNewRaw	Set the Source-Port, input expects raw value
LatencyFlowIdRaw	Change the latency flow-Id, input expects raw value
MirrorIngressRaw	Set mirror in ingress, input expects raw value
OamRaw	Changes 4 OAM signals (OAM-Stamp-Offset, OAM-offset, OAM-Sub-Type, OAM-Up-Mep), input expects raw value
TrapRaw	Set Trap code, input should be raw value. At egress stage refers only to the TRAP ID (strength is not relevant, and can be changed using bcmFieldActionTrapStrength action)
StatProfile0Raw	Updates statistics profile 0. Input expects raw value
StatProfile1Raw	Updates statistics profile 1. Input expects raw value
StatProfile2Raw	Updates statistics profile 2. Input expects raw value
StatProfile3Raw	Updates statistics profile 3. Input expects raw value
StatProfile4Raw	Updates statistics profile 4. Input expects raw value
StatProfile5Raw	Updates statistics profile 5. Input expects raw value
StatProfile6Raw	Updates statistics profile 6. Input expects raw value
StatProfile7Raw	Updates statistics profile 7. Input expects raw value
StatProfile8Raw	Updates statistics profile 8. Input expects raw value
StatProfile9Raw	Updates statistics profile 9. Input expects raw value
StatSamplingRaw	Update statistical sampling value, input expects raw value
StartPacketStripRaw	Strip the start of packet. Refers to bcm_field_start_packet_strip_t. Input expects raw value
ParsingStartTypeRaw	Indicate the layer type that egress parser should consider, input expects raw value
ParsingStartOffsetRaw	Indicate from which offset egress parser should start parsing (all info before that will not be parsed). Input expects raw value
IPTCommandRaw	IPT Profile command. Input expects raw value
LearnRaw0	Updates learn info 0
LearnRaw1	Updates learn info 1
LearnRaw2	Updates learn info 2
LearnRaw3	Updates learn info 3
LearnRaw4	Updates learn info 4
StatOamLMRaw	Update the counter of OAM LM with a raw value
LearnKey0	Updates bits 31:0 of learn info key
LearnKey1	Updates bits 63:32 of learn info key
LearnKey2	Updates learn info key and app_db_index. value[0]: bits 79:64 of learn info ley, value[1] app_db_index
LearnKey2Raw	Updates learn info key and app_db_index. Key is at the LSB

Table 52: iPMF-3 Actions (Continued)

bcmFieldAction…	Description
LearnPayload0	Updates bits 31:0 of learn info data (payload)
LearnPayload1	Updates bits 63:32 of learn info data (payload)
LearnEntryFormatIndex	Updates entry format index in learn info
LearnOrTransplant	Updates learn or transplant bit

Table 53: Egress-PMF Actions

bcmFieldAction…	Description
PrioIntNew	TC update **NOTE:** In the ePMF, setting MC traffic to TC=4 is not allowed and can cause traffic issues. Do not use this setting in that case.
Drop	Discard packet
MirrorEgress	Update the mirror profile, mirror_cmd should be encoded as MIRROR Gport
DropPrecedence	Set Packet's Drop Precedence, the values are BCM_FIELD_COLOR_XXX
Trap	Set Trap code, use TRAP GPORT type value[0]=TRAP_GPORT value[1]=Trap Qualifier. at egress stage refers only to the TRAP ID (strength is not relevant, and can be changed using bcmFieldActionTrapStrength action)
Snoop	Snoop matched packets, input can be either a snoop command: value[0] = Mirror Gport, value[1]=snoop qualifier, value[2]=snoop strength OR: snoop code: value[0] = Trap Gport, value[1]=snoop qualifier, at egress stage refers only to the snoop ID (strength is not relevant, and can be changed using bcmFieldActionSnoopStrength action)
QosMapIdNew	Set the QoS profile
Void	Void action for cascading, saving buffer in the action payload (for FEM use)
StatId0	Updates statistics ID 0
StatId1	Updates statistics ID 1
StatProfile0	Updates statistics profile 0. value[0]: is_meter. value[1]: is_lm. value[2]: type. value[3]: valid.
StatProfile1	Updates statistics profile 1. value[0]: is_meter. value[1]: is_lm. value[2]: type. value[3]: valid.
InvalidNext	Invalid next Action macro Id
AceEntryId	Pointer to the ACE entry ID
TrapStrength	Updates the Trap Strength. Use TRAP Gport
SnoopStrength	updates the snoop strength
OutPortTrafficManagement	Updates the Out-TM port
SnoopRaw	Snoop matched packets, input should be raw value. At egress stage refers only to the snoop ID (strength is not relevant, and can be changed using bcmFieldActionSnoopStrength action)
DropPrecedenceRaw	Set Packet's Drop Precedence, input expects raw value.
MirrorEgressRaw	Update the egress mirror profile, input expects raw value.
TrapStrengthRaw	Updates the Trap Strength, input expects raw value.
SnoopStrengthRaw	Updates the Snoop Strength, input expects raw value.

Table 54: Egress-Extension (ACE) PMF Actions

bcmFieldAction…	Description
PrioIntNew	TC update
EcnNew	Change metadata ECN or congestion information (including CNI)
DropPrecedence	Set Packet's Drop Precedence, the values are BCM_FIELD_COLOR_XXX

Table 54: Egress-Extension (ACE) PMF Actions (Continued)

bcmFieldAction…	Description
Stat0	Stat meter object. value[0]: stat_meter_obj_cmd. value[1]: stat_meter_obj_id
Stat1	Stat counter object. value[0]: stat_counter_obj_cmd. value[1]: stat_counter_obj_id. value[2]: valid
TtlSet	New TTL
LearnSrcPortNew	Replace the learnt packet source Port
InterfaceClassVPort	Set InLIF profile
Void	Void action, used for cascading, saving buffer in the action payload (for FEM use)
IPTProfile	IPT Profile action (used for instrumentation)
InvalidNext	Invalid next Action macro Id
OutVport0	Change OutLIF 0, value should be encoded as a GPORT with LIF subtype
OutVport1	Change OutLIF 1, value should be encoded as a GPORT with LIF subtype
OutVport2	Change OutLIF 2, value should be encoded as a GPORT with LIF subtype
OutVport3	Change OutLIF 3, value should be encoded as a GPORT with LIF subtype
ForwardingLayerIndex	Update forwarding layer index
OutInterface0	Change OutLIF 0, value should be encoded as type interface
OutInterface1	Change OutLIF 1, value should be encoded as type interface
OutInterface2	Change OutLIF 2, value should be encoded as type interface
OutInterface3	Change OutLIF 3, value should be encoded as type interface
AceContextValue	Updates programmable value per ACE context, bcm_field_ace_context_t. Up to four different values can be given in value[0-3], although some may be mutually exclusive
ForwardingAdditionalInfo	Updates forwarding additional information
OutVport0Raw	Change the OutLIF 0, with raw value
OutVport1Raw	Change the OutLIF 1, with raw value
OutVport2Raw	Change the OutLIF 2, with raw value
OutVport3Raw	Change the OutLIF 3, with raw value
DropPrecedenceRaw	Set Packet's Drop Precedence, input expects raw value
AceContextValueRaw	Updates programmable value per ACE context, input expects raw value
Stat0Raw	Stat meter object. Input expects raw value
Stat1Raw	Stat counter object. Input expects raw value

NOTE: Action sizes can be obtained by using the following shell command:

```
field map action bcm <stage=ipmf1/2/3..epmf>
```

11.34 Information Specific to the BCM88830 (Jericho2X) Device

The following table shows the number of various resources per stage.

Resource	Stage			
	iPMF-1	**iPMF-2**	**iPMF-3**	**ePMF**
Context selection lines	128	128	128	128
Contexts	64		64	64
Total number of keys	5 + 5 initial keys	5	3	4
TCAM lookup keys	4	4	2	2
Direct extraction keys	0	2	1	1
Exact match keys	1	1		1
FES contexts	32		16	12
FEM	16		0	0

Table 55: Database Properties

Type	TCAM Lookup
Maximum key size	Possible key sizes
	80b/160b/320b
Maximum payload size:	32b/64b/128b
Memory	Shared (with other TCAM FGs)
Maximum number of entries per FG	50K/25K/12.5K
Supported stages	iPMF1/2/3 ePMF
Maximum FGs (in a given stage) per context:	4/4/2 – iPMF1/2 2/2/1 – iPMF3 2/2/1 – ePMF **NOTE:** For ePMF, double lookup key size must be > 160b.

11.34.1 CS Qualifiers

For a list of Context Selection Qualifiers, see Section 11.31.1, CS Qualifiers.

To see latest version support, run the following diagnostic command:

```
field qualifier ContextSelection
```

Table 56: List of iPMF1 Stage Context Selection Qualifiers

bcmFieldQualify…	Description
ForwardingType	Use this qualifier together with `input_arg` to indicate which layer number relative to forwarding layer to qualify upon. Possible layer types are bcmFieldLayerTypeXXX.
VlanFormat	Incoming VLAN structure format (untagged/single/double/priority-tagged). For possible values, see bcm_port_tag_struct_type_t.

Table 56: List of iPMF1 Stage Context Selection Qualifiers (Continued)

bcmFieldQualify…	Description
RxTrapCode	Qualify on RX Trap Code
Ptch	Opaque attribute field of the Injected packets. Part of the PTCH_2 header. The input expects a raw value.
VPortRangeCheck	Virtual Port Range Check Results
AppType	Packet Application type used by the ACL lookup. Can be either predefined (`bcm_field_AppType_t`) or user-defined.
ForwardingLayerIndex	Qualify on the Forwarding Layer Index
ForwardingLayerQualifier	Use this qualifier together with `input_arg` to indicate which layer number relative to forwarding layer to qualify upon. It qualifies Layer Qualifiers (look for layer record qualifiers in UM).
TracePacket	Qualifies upon Packets that Trace was set for them
InVportClass	Qualify on the profile of an in LIF encoded as Gport
PortClassPacketProcessing	PP port profile
PortClassTrafficManagement	TM port profile
AppTypePredefined	The predefined AppType (bcm_field_AppType_t) used by the forwarding lookup.
VlanFormatRaw	Qualifies on the incoming VLAN structure format (untagged, single tagged, double-tagged, or priority-tagged). The input expects a raw value.
AppTypeRaw	Qualifies on packet Application type used by the ACL lookup. Can be either predefined or user-defined. Input expects raw value.
AppTypePredefinedRaw	Qualifies on the predefined AppType used by the forwarding lookup. Input expects raw value.
ForwardingTypeRaw	The same qualifier as bcmFieldQualifyForwardingType without the mapping function. That is, the input expects a raw value (HW value).
PrtQualifier	Qualifies on the prt_qualifier signal, which may be set by the injected packet (Opaque attribute field part of the PTCH_2 header). The input expects an enum value `bcm_field_prt_qualifier_t`. Similar to the ptch qualifier, which uses a raw value and not an enum.
AcInLifWideData	Qualifies on the Ac InLIF wide (generic) data.

Table 57: List of iPMF2 Stage Context Selection Qualifiers

bcmFieldQualify…	Description
CascadedKeyValue	Value of key cascaded from prior group in cascade
CompareKeysResult0	Compare keys result 0
CompareKeysResult1	Compare keys result 1
StateTableDataWrite	State table payload, as written to state table

Table 58: List of iPMF3 Stage Context Selection Qualifiers

bcmFieldQualify…	Description
ForwardingType	Qualify upon Forwarding Layer Type. Possible layer types are bcmFieldLayerTypeXXX
VPortRangeCheck	Virtual Port Range Check Results
ContextId	Qualifies on the Context Id (see bcmFieldForwardContextXXX)
ForwardingLayerQualifier	Use this qualifier together with `input_arg` to indicate which layer number relative to forwarding layer to qualify upon. It qualifies Layer Qualifiers (look for layer record qualifiers in UM).
RpfEcmpMode	RPF ECMP mode
PortClassPacketProcessing	PP port profile

Table 58: List of iPMF3 Stage Context Selection Qualifiers (Continued)

bcmFieldQualify…	Description
PortClassTrafficManagement	TM port profile
ForwardingTypeRaw	The same qualifier as bcmFieldQualifyForwardingType without the mapping function. That is, the input expects a raw value (HW value).

Table 59: List of Egress-PMF Stage Context Selection Qualifiers

bcmFieldQualify…	Description
SrcClassField	PPH Value 2
DstClassField	PPH Value 1
ForwardingType	Use this qualifier together with `input_arg` to indicate which layer number relative to forwarding layer to qualify upon. Possible layer types are bcmFieldLayerTypeXXX.
VlanFormat	Incoming vlan structure format (untagged/single/double/priority-tagged). For possible values, see bcm_port_tag_struct_type_t.
FheiSize	DNX FHEI header size in bytes
PphPresent	Qualifies on PPH present field in ITMH
UDHBase0	User-defined Header 0 encoded type and size – Value 1: type PMF size 32b
UDHBase1	User-defined Header 1 encoded type and size – Value 1: type PMF size 32b
UDHBase2	User-defined Header 2 encoded type and size – Value 1: type PMF size 32b
UDHBase3	User-defined Header 3 encoded type and size – Value 1: type PMF size 32b
ContextId	Qualifies on the Context ID (see bcmFieldForwardContextXXX).
ForwardingLayerQualifier	Use this qualifier together with `input_arg` to indicate which layer number relative to forwarding layer to qualify upon. It qualifies Layer Qualifiers (look for layer record qualifiers in UM).
PortClassPacketProcessing	PP port profile
InVportClass0	InLIF Profile 0, by Gport
VlanFormatRaw	Qualifies on the incoming VLAN structure format (untagged, single tagged, double-tagged, or priority-tagged). The input expects a raw value
LearnExtensionPresent	Qualifies on whether PPH learn extension is present
ForwardingTypeRaw	The same qualifier as bcmFieldQualifyForwardingType without the mapping function. That is, the input expects a raw value (HW value).
ContextIdRaw	Qualifies on the Forwarding Context value. Input expects raw value. Relevant for egress stage only
FtmhAsePresent	Qualify packet on FTMH ASE present field
FtmhAseType	Qualify packet on FTMH ASE type value

11.34.2 Qualifiers List

To see latest version support, run the following diagnostic command:

```
field qualifier Predefined bcm stage=iPMF1/2/3/ePMF/external class=meta/layer/header
```

11.34.2.1 Header Qualifiers List

Table 60: Header Qualifiers List

bcmFieldQualify…	Description
SrcIp6	IPv6 Source address
DstIp6	IPv6 destination address
SrcIp6High	Source IPv6 Address (High/Upper 64 bits, 64 MSB)
DstIp6High	Destination IPv6 Address (High/Upper 64 bits, 64 MSB)
SrcIp6Low	Source IPv6 Address (Low/Lower 64 bits, 64 LSB)
DstIp6Low	Destination IPv6 Address (Low/Lower 64 bits, 64 LSB)
SrcMac	Source MAC address
DstMac	Destination MAC address
SrcIp	Packet Source IP address
DstIp	Packet destination IP address
Ip6FlowLabel	IPv6 Flow Label
L4SrcPort	L4 source port
L4DstPort	L4 destination port
Ip6NextHeader	Next protocol field in IPv6 Header
Ip6TrafficClass	Traffic class field in IPv6 Header
Ip6HopLimit	Hop Count field in IPv6 Header
TcpControl	TCP control Flags
IpFlags	IP Flags Field
ExtensionHeaderType	Qualifies on Next Header Field in First Extension Header
ArpSenderIp4	Sender IPv4 field of ARP header
ArpTargetIp4	Target IPv4 field of ARP header
ArpOpcode	Opcode field of ARP header
Ip4Protocol	Qualify Upon IPv4 Protocol
Ip4Tos	Qualify Upon IPv4 TOS
Ip4Ttl	Qualify Upon IPv4 TTL
EtherTypeUntagged	802.1 Ethernet Type, only for untagged frames
Tpid	VLAN TPID, offset of the value is inside the VLAN tag. Need to add the offset where the tag start in the Layer (inner/outer).
VlanId	VLAN ID, offset of the value is inside the VLAN tag. Need to add the offset where the tag start in the Layer (inner/outer).
VlanPri	VLAN P-bits, offset of the value is inside the VLAN tag. Need to add the offset where the tag start in the Layer (inner/outer).
VlanCfi	VLAN CFI, offset of the value is inside the VLAN tag. Need to add the offset where the tag start in the Layer (inner/outer).
VlanPriCfi	VLAN P-bits and CFI, offset of the value is inside the VLAN tag. Need to add the offset where the tag start in the Layer (inner/outer).
MplsLabel	MPLS Label

Table 60: Header Qualifiers List (Continued)

bcmFieldQualify…	Description
MplsLabelId	ID field of MPLS Label
MplsLabelTtl	TTL field of MPLS Label
MplsLabelBos	BOS field of MPLS Label
MplsLabelExp	TTL field of MPLS Label

11.34.2.2 Layer Record Qualifiers List

Table 61: Layer Record Qualifiers List

bcmFieldQualify…	Description
IpFrag	IP fragments
LayerRecordType	Qualifies on the Layer Record Type. For possible values look at bcm_field_layer_type_t
LayerRecordOffset	Qualifies on the Layer Record Offset
LayerRecordQualifier	Qualifies on the Layer Record qualifier
EthernetBroadcast	Qualify on Ethernet packets for which the destination MAC address is broadcast (MAC-DA[47:0] is all 1)
EthernetFirstTpidExist	Qualify on Ethernet packets for which their first VLAN TPID exists
EthernetFirstTpidIndex	Qualify on Ethernet packets first VLAN TPID Index
EthernetSecondTpidExist	Qualify on Ethernet packets for which their second VLAN TPID exists
EthernetSecondTpidIndex	Qualify on Ethernet packets' second VLAN TPID Index
EthernetThirdTpidExist	Qualify on Ethernet packets for which their third VLAN TPID exists
EthernetThirdTpidIndex	Qualify on Ethernet packets' third VLAN TPID Index
L2Format	Qualify on the type of the frame (see bcmFieldL2FormatXXX)
IpHasOptions	Qualify on IPv4 packets where the IP Header include options
IpFirstFrag	Qualify on IPv4 packets where the IP Header is fragmented and this is the first fragment
IpTunnelType	Qualify on IPv4 TunnelType (see bcmFieldTunnelXXX)
Ip6MulticastCompatible	Qualify on IPv6 packets for which their DIP is Multicast (8 MSBs are 0xFF)
Ip6FirstAdditionalHeaderExist	Qualify on IPv6 packets' first additional header exists
Ip6FirstAdditionalHeader	Qualify on IPv6 packets' first additional header (see bcmFieldIp6AdditionalHeaderXXX)
Ip6SecondAdditionalHeaderExist	Qualify on IPv6 packets' second additional header exists
Ip6SecondAdditionalHeader	Qualify on IPv6 packets second additional header (see bcmFieldIp6AdditionalHeaderXXX)
ItmhPphType	Qualify on Ingress TM Header PPH type
LayerRecordTypeRaw	Qualifies on the Layer Record Type. For possible values, look at bcm_field_layer_type_t; input expects a raw value.
Ip4DstMulticast	Qualify on IPv4 packets for which their DIP is multicast
IpTunnelTypeRaw	Raw Qualify on IPv4 TunnelType

11.34.2.3 Metadata Qualifiers List

Table 62: List of iPMF-1,2 Qualifiers

bcmFieldQualify…	Description
InPort	Ingress Port, value should be encoded as LOCAL GPORT Type. Includes core ID, and can be used by entries for all cores
L4PortRangeCheck	Qualify on L4 Ops (L4 source and dest ports) range. The qualifier is a bitmap and not specific range id, it means if range 2 was configured and the qualifier is hit, the entry value is expected to be 0x100
SrcPort	Qualify on source port, value should be encoded as SYS_PORT GPORT Type
DstPort	Qualify on destination port, all ingress stages expect raw value, egress expects GPORT encoded as SYS_PORT
PacketRes	When TRUE packet lookup result was Unknown Destination
InterfaceClassL2	VSI Profile
Vrf	VRF Id
VlanFormat	Incoming vlan structure format (untagged/single/double/priority-tagged). For possible values, see bcm_port_tag_struct_type_t.
DstMulticastGroup	Multicast Group id, encoded as GPORT
SrcModPortGport	Qualify on Source module/port pair, value should be encoded as SYS_PORT GPORT Type
Color	Qualify based on packet color, value should be one of BCM_FIELD_COLOR_XXX
IntPriority	Internal priority (TC)
Vpn	VSI Id
MplsForwardingLabelAction	MPLS forwarding label action
L2Learn	L2 learn enable
EcnValue	Qualify on metadata ECN or congestion information (including CNI)
RxTrapCode	Qualify on RX Trap Code
Ptch	Opaque attribute field of the Injected packets. Part of the PTCH_2 header. The Input expects a raw value.
RxTrapData	Qualify on RX Trap qualifier
TrillEgressRbridge	Egress RBridge Nickname
ISid	I-SID (MAC-in-MAC lookup-id)
DstRpfGport	RPF destination (gport) for the RPF check
TrunkHashResult	Trunk Hash Result (i.e., the Load-balancing Key)
PacketLengthRangeCheck	Packet length range. The range ID on which the packet is matched. If the packet size > 144B, it is treated as 144B.
LearnSrcMac	Qualifies on learn source MAC
LearnVlan	Qualifies on learn VLAN
PacketSize	Packet size in Bytes
TranslatedOuterVlanId	Translated Outer VLAN ID
TranslatedOuterVlanPri	Translated Outer VLAN priority
TranslatedOuterVlanCfi	Translated Outer VLAN CFI
TranslatedInnerVlanId	Translated inner VLAN Id
TranslatedInnerVlanPri	Translated inner VLAN priority
TranslatedInnerVlanCfi	Translated inner VLAN CFI
RxTrapCodeForSnoop	Qualify on Snoop Code
IncomingIpIfClass	RIF profile
RxTrapStrength	Qualify on RX Trap Strength

Table 62: List of iPMF-1,2 Qualifiers (Continued)

bcmFieldQualify…	Description
OamUpMep	Indicates if the OAM packet is vUP-MEP (sent to a destination in the network, as opposed to a specific port)
OamSubtype	In OAM the packet type is specified in the OAM header and mapped to a subtype in the hardware
OamHeaderOffset	Indicates the offset of the OAM header relative to the start of packet (as opposed to start of header-offset)
OamStampOffset	Indicates the offset to the position, in the OAM header, where the ToD or counter value should be stamped relative to the start of packet
OamMepId	If MEP is handled in OAMP, then the OAM-ID is the MEP-ID (equivalent to the index used to access the MEP DB)
OamMeterDisable	Attribute that is passed to the PMF and can also be configured by the user per MEP
VlanAction	VLAN translation action ID
GeneratedTtl	Qualify on Packet's generate Time-To-Live
IpMulticastCompatible	Packet is compatible for multicast
PacketIsIEEE1588	Indicating whether the packet is 1588
IEEE1588Encapsulation	IEEE-1588 Encapsulation according to bcm_field_IEEE1588Encap_t
IEEE1588CompensateTimeStamp	IEEE-1588 update time stamp
IEEE1588Command	Command used by egress pipeline, indicating if CF(correction field) needs to be update
IEEE1588HeaderOffset	This field indicates the offset in bytes of the IEEE-1588 header
KeyGenVar	Match on configured key Gen Variable (values that was configured for context)
NetworkQos	Qualify Upon Network Qos
EcmpLoadBalanceKey0	ECMP Load Balance Key 0
EcmpLoadBalanceKey1	ECMP Load Balance Key 1
EcmpLoadBalanceKey2	ECMP Load Balance Key 2
ForwardingLayerIndex	Qualify on the Forwarding Layer Index
AcInLifWideData	Qualifies on the Ac InLIF wide (generic) data
NativeAcInLifWideData	Qualifies on the native Ac InLIF wide (generic) data
TracePacket	Qualifies upon Packets that Trace was set for them
InInterface	Ingress Logical Interface (LIF/RIF), value should be encoded as L3 interface
OutInterface	Egress Logical Interface (LIF/RIF), should be encoded as L3 interface
PortClassPacketProcessing	PP port profile
PortClassTrafficManagement	TM port profile
InVPort0	In LIF 0, value should be GPORT with subtype LIF
InVPort1	In LIF 1, value should be GPORT with subtype LIF
OutVPort0	OutLIF 0, value should be GPORT with subtype LIF
OutVPort1	OutLIF 1, value should be GPORT with subtype LIF
BierStringOffset	Qualifies upon Bier string offset
BierStringSize	Qualifies upon Bier string size
PacketIsBier	Qualifies upon Bier packets
PortClassPacketProcessingGeneralData	PP port general data
ForwardingAdditionalInfo	Qualifies on the Forwarding additional info
InVportClass0	InLIF Profile 0, by Gport
InVportClass1	InLIF Profile 1, by Gport
StatId0	Qualifies on statistics ID 0

Table 62: List of iPMF-1,2 Qualifiers (Continued)

bcmFieldQualify…	Description
StatId1	Qualifies on statistics ID 1
StatId2	Qualifies on statistics ID 2
StatId3	Qualifies on statistics ID 3
StatId4	Qualifies on statistics ID 4
StatId5	Qualifies on statistics ID 5
StatId6	Qualifies on statistics ID 6
StatId7	Qualifies on statistics ID 7
StatId8	Qualifies on statistics ID 8
StatId9	Qualifies on statistics ID 9
StatProfile0	Qualifies on statistics profile 0
StatProfile1	Qualifies on statistics profile 1
StatProfile2	Qualifies on statistics profile 2
StatProfile3	Qualifies on statistics profile 3
StatProfile4	Qualifies on statistics profile 4
StatProfile5	Qualifies on statistics profile 5
StatProfile6	Qualifies on statistics profile 6
StatProfile7	Qualifies on statistics profile 7
StatProfile8	Qualifies on statistics profile 8
StatProfile9	Qualifies on statistics profile 9
StatMetaData	Qualifies on Statistics metadata
InVPort0Raw	Qualifies on In LIF 0 input expects raw value
InVPort1Raw	Qualifies on In LIF 1 input expects raw value
OutVPort0Raw	Qualifies on OutLIF 0 input expects raw value
OutVPort1Raw	Qualifies on OutLIF 1 input expects raw value
MirrorCode	Matches on Mirror Code
LearnVsi	Qualifies on learn VSI
LearnData	Qualifies on data field in the Learn Info
LearnStationMove	Qualifies on learn Station move
LearnMatch	Qualifies on learn Match
LearnFound	Qualifies on learn Found
LearnExpectedWon	Qualifies on learn Expected Won
VrfValue	Qualifies on the VRF part of forwarding data, without checking that the type is VRF
InPortWithoutCore	Ingress Port, value should be encoded as LOCAL GPORT Type. Does not include the core ID, caution should be taken when not adding entry by core
SaLookupAcceptedStrength	Qualifies on Accepted strength LSB returned from SA Lookup result. Qualifier is only relevant for bridged packets
SrcPortRaw	Qualifies on source port, input expects raw value
InPortWithoutCoreRaw	Qualifies on Ingress Port. Does not include the core ID, caution should be taken when not adding entry by core, input expects raw value
VlanFormatRaw	Qualifies on the incoming VLAN structure format (untagged, single tagged, double-tagged, or priority-tagged). The input expects a raw value
InPortRaw	Qualifies on Ingress Port. Includes core ID, and can be used by entries for all cores, input expects raw value
ColorRaw	Qualifies on packet color, input expects raw value

Table 62: List of iPMF-1,2 Qualifiers (Continued)

bcmFieldQualify…	Description
DstRpfValid	Checks if the RPF Destination is valid
RpfOutVPort	Qualifies on the RPF OUT LIF. Value should be GPORT with subtype LIF
RpfOutInterface	Qualifies on the RPF OUT LIF. Value should be L3 interface
RpfOutVPortRaw	Qualifies on the RPF OUT LIF
RpfRouteValid	Qualifies on the RPF Route valid
RangeFirstHit0	Qualifies on L4 Ops Extended Encoder0. Encoder0 can be configured to consider a certain subset of range types, so for the chosen subset, the encoder will output a maximum of 8b that consists of 1 valid bit (indicates if at least one hit occurred inside the subset of chosen ranges) and 7b of the first hit index. The valid 1b offset is dynamically allocated according to the number of chosen range types in the subset (that is, the offset is 6 when one type is selected, 7 when two to three types are selected, and 8 when four types are selected).
RangeFirstHit1	See bcmFieldQualifyRangeFirstHit0
RangeFirstHit2	See bcmFieldQualifyRangeFirstHit0
RangeFirstHit3	See bcmFieldQualifyRangeFirstHit0
VipValid	Qualifies on SLLB Virtual Wire VW_VIP_VALID
VipId	Qualifies on SLLB Virtual Wire VW_VIP_ID
VIPMemberReference	Qualifies on SLLB Virtual Wire VW_MEMBER_REFERENCE
PccHit	Qualifies on SLLB Virtual Wire VW_PCC_HIT
PrtQualifier	Qualifies on the prt_qualifier signal, which may be set by the injected packet (Opaque attribute field part of the PTCH_2 header). The input expects an enum value `bcm_field_prt_qualifier_t`. Similar to the ptch qualifier, which uses a raw value and not an enum.

Table 63: List of iPMF-3 Qualifiers

bcmFieldQualify…	Description
InPort	Ingress Port, value should be encoded as LOCAL GPORT Type. Includes core ID, and can be used by entries for all cores
SrcPort	Qualify on source port, value should be encoded as SYS_PORT GPORT Type
DstPort	Qualify on destination port, all ingress stages expect raw value, egress expects GPORT encoded as SYS_PORT
InterfaceClassL2	VSI Profile
Vrf	VRF Id
SrcModPortGport	Qualify on Source module/port pair, value should be encoded as SYS_PORT GPORT Type
Color	Qualify based on packet color, value should be one of BCM_FIELD_COLOR_XXX
IntPriority	Internal priority (TC)
Vpn	VSI Id
FlowId	Qualify on destination Flow-Id
MplsForwardingLabelAction	MPLS forwarding label action
L2Learn	L2 learn enable
EcnValue	Qualify on metadata ECN or congestion information (including CNI)
RxTrapCode	Qualify on RX Trap Code
RxTrapData	Qualify on RX Trap qualifier

Table 63: List of iPMF-3 Qualifiers (Continued)

bcmFieldQualify…	Description
TrillEgressRbridge	Egress RBridge Nickname
ISid	I-SID (MAC-in-MAC lookup-id)
DstRpfGport	RPF destination (gport) for the RPF check
TrunkHashResult	Trunk Hash Result (i.e., the Load-balancing Key)
LearnSrcMac	Qualifies on learn source MAC
LearnVlan	Qualifies on learn VLAN
TranslatedOuterVlanId	Translated Outer VLAN ID
TranslatedOuterVlanPri	Translated Outer VLAN priority
TranslatedOuterVlanCfi	Translated Outer VLAN CFI
TranslatedInnerVlanId	Translated inner VLAN Id
TranslatedInnerVlanPri	Translated inner VLAN priority
TranslatedInnerVlanCfi	Translated inner VLAN CFI
DstGport	Qualifies packet according to destination encoded as trap code
RxTrapCodeForSnoop	Qualify on Snoop Code
IncomingIpIfClass	RIF profile
StackingRoute	Stacking Route History bitmap
RxTrapStrength	Qualify on RX Trap Strength
OamUpMep	Indicates if the OAM packet is vUP-MEP (sent to a destination in the network, as opposed to a specific port)
OamSubtype	In OAM the packet type is specified in the OAM header and mapped to a subtype in the hardware
OamHeaderOffset	Indicates the offset of the OAM header relative to the start of packet (as opposed to start of header-offset)
OamStampOffset	Indicates the offset to the position, in the OAM header, where the ToD or counter value should be stamped relative to the start of packet
OamMepId	If MEP is handled in OAMP, then the OAM-ID is the MEP-ID (equivalent to the index used to access the MEP DB)
VlanAction	VLAN translation action ID
GeneratedTtl	Qualify on Packet's generate Time-To-Live
PacketIsIEEE1588	Indicating whether the packet is 1588
IEEE1588Encapsulation	IEEE-1588 Encapsulation according to bcm_field_IEEE1588Encap_t
IEEE1588CompensateTimeStamp	IEEE-1588 update time stamp
IEEE1588Command	Command used by egress pipeline, indicating if CF(correction field) needs to be update
IEEE1588HeaderOffset	This field indicates the offset in bytes of the IEEE-1588 header
KeyGenVar	Match on configured key Gen Variable (values that was configured for context)
Container	This qualifier will be used as container in IPMF3, to receive the action buffer, when performing cascading between IPMF1/2 and IPMF3
UDHBase0	User-defined Header 0 encoded type and size – Value 1: type PMF size 32b
UDHBase1	User-defined Header 1 encoded type and size – Value 1: type PMF size 32b
UDHBase2	User-defined Header 2 encoded type and size – Value 1: type PMF size 32b
UDHBase3	User-defined Header 3 encoded type and size – Value 1: type PMF size 32b
NetworkQos	Qualify Upon Network Qos
ForwardingLayerIndex	Qualify on the forwarding layer index
IPTProfile	Qualifies upon IPT Profile (End Of Packet Editing)

Table 63: List of iPMF-3 Qualifiers (Continued)

bcmFieldQualify…	Description
UDHData0	User-defined Header 0 payload MS bits of UDH
UDHData1	User-defined Header 1 payload
UDHData2	User-defined Header 2 payload
UDHData3	User-defined Header 3 payload LS bits of UDH
RxSnoopStrength	Snoop Strength
StatSamplingCode	Statistical Sampling Code value
StatSamplingQualifier	Statistical Sampling qualifier value
RpfEcmpMode	RPF ECMP mode
StatOamLM	OAM LM counter value, the index of the values are bcmFieldStatOamLmIndexXXX
InInterface	Ingress Logical Interface (LIF/RIF), value should be encoded as L3 interface
OutInterface	Egress Logical Interface (LIF/RIF), should be encoded as L3 interface
PortClassPacketProcessing	PP port profile
PortClassTrafficManagement	TM port profile
InVPort0	In LIF 0, value should be GPORT with subtype LIF
InVPort1	In LIF 1, value should be GPORT with subtype LIF
OutVPort0	OutLIF 0, value should be GPORT with subtype LIF
OutVPort1	OutLIF 1, value should be GPORT with subtype LIF
OutVPort2	OutLIF 2, value should be GPORT with subtype LIF
OutVPort3	OutLIF 3, value should be GPORT with subtype LIF
BierStringOffset	Qualifies upon Bier string offset
BierStringSize	Qualifies upon Bier string size
PacketIsBier	Qualifies upon Bier packets
PortClassPacketProcessingGeneralData	PP port general data
ForwardingAdditionalInfo	Qualifies on the Forwarding additional info
InVportClass0	InLIF Profile 0, by Gport
InVportClass1	InLIF Profile 1, by Gport
StatMetaData	Qualifies on Statistics metadata
InVPort0Raw	Qualifies on In LIF 0 input expects raw value
InVPort1Raw	Qualifies on In LIF 1 input expects raw value
OutVPort0Raw	Qualifies on OutLIF 0 input expects raw value
OutVPort1Raw	Qualifies on OutLIF 1 input expects raw value
OutVPort2Raw	Qualifies on OutLIF 2 input expects raw value
OutVPort3Raw	Qualifies on OutLIF 3 input expects raw value
MirrorCode	Matches on Mirror Code
MirrorData	Matches on Mirror Data (qualifier)
LearnVsi	Qualifies on learn VSI
LearnData	Qualifies on data field in the Learn Info
LearnStationMove	Qualifies on learn Station move
LearnMatch	Qualifies on learn Match
LearnFound	Qualifies on learn Found
LearnExpectedWon	Qualifies on learn Expected Won
VrfValue	Qualifies on the VRF part of forwarding data, without checking that the type is VRF

Table 63: List of iPMF-3 Qualifiers (Continued)

bcmFieldQualify…	Description
InPortWithoutCore	Ingress Port, value should be encoded as LOCAL GPORT Type. Does not include the core ID, caution should be taken when not adding entry by core
SrcPortRaw	Qualifies on source port, input expects raw value
InPortWithoutCoreRaw	Qualifies on Ingress Port. Does not include the core ID, caution should be taken when not adding entry by core, input expects raw value
InPortRaw	Qualifies on Ingress Port. Includes core ID, and can be used by entries for all cores, input expects raw value
ColorRaw	Qualifies on packet color, input expects raw value
DstRpfValid	Checks if the RPF Destination is valid
EcmpGroup	Qualifies on the ECMP group
StatOamLMRaw	OAM LM counter raw value

Table 64: List of Egress-PMF Qualifiers

bcmFieldQualify…	Description
L4PortRangeCheck	Qualify on L4 Ops (L4 source and dest ports) range. The qualifier is a bitmap and not specific range id, it means if range 2 was configured and the qualifier is hit, the entry value is expected to be 0x100
SrcPort	Qualify on source port, value should be encoded as SYS_PORT GPORT Type
DstPort	Qualify on destination port, all ingress stages expect raw value, egress expects GPORT encoded as SYS_PORT
SrcClassField	PPH Value 2
DstClassField	PPH Value 1
InterfaceClassL2	VSI Profile
Vrf	VRF Id
OutPort	Egress port, value should be encoded as LOCAL GPORT Type. Does not include the core ID, caution should be taken when not adding entry by core
DstL3Egress	L3 Egress Interface, Raw value of SYS_PORT
DstMulticastGroup	Multicast Group id, encoded as GPORT
SrcModPortGport	Qualify on Source module/port pair, value should be encoded as SYS_PORT GPORT Type
Color	Qualify based on packet color, value should be one of BCM_FIELD_COLOR_XXX
IntPriority	Internal priority (TC)
Vpn	VSI Id
RepCopy	Qualify on Copy type: 0:Forwarded / 1:Snooped / 2:Mirrored / 3:Statistical Sampling, packet
EcnValue	Qualify on metadata ECN / congestion information (including CNI)
RxTrapCode	Qualify on RX Trap Code
TrunkHashResult	Trunk Hash Result (i.e., the Load-balancing Key)
LearnSrcMac	Qualifies on learn source MAC
LearnVlan	Qualifies on learn VLAN
Fhei	DNX FHEI header field
FheiSize	DNX FHEI header size in bytes
StackingRoute	Stacking Route History bitmap
RxTrapStrength	Qualify on RX Trap Strength
OamTsSystemHeader	FTMH Application specific extension

Table 64: List of Egress-PMF Qualifiers (Continued)

bcmFieldQualify…	Description
DstSysPortExt	Qualifies upon TM Destination extension header
GeneratedTtl	Qualify on Packet's generate Time-To-Live
PphPresent	Qualifies on PPH present field in ITMH
PacketProcessingInVportClass	Qualifies on InLIF profile
UDHBase0	User-defined Header 0 encoded type and size – Value 1: type PMF size 32b
UDHBase1	User-defined Header 1 encoded type and size – Value 1: type PMF size 32b
UDHBase2	User-defined Header 2 encoded type and size – Value 1: type PMF size 32b
UDHBase3	User-defined Header 3 encoded type and size – Value 1: type PMF size 32b
NetworkQos	Qualify Upon Network Qos
AceEntryId	Qualify Upon Ace Entry ID
NetworkLoadBalanceKey	Network Load Balance Key
IPTProfile	Qualifies upon IPT Profile (End Of Packet Editing)
UDHData0	User-defined Header 0 payload MS bits of UDH
UDHData1	User-defined Header 1 payload
UDHData2	User-defined Header 2 payload
UDHData3	User-defined Header 3 payload LS bits of UDH
RxSnoopStrength	Snoop Strength
RxSnoopCode	Qualify upon Snoop profile
InInterface	Ingress Logical Interface (LIF/RIF), value should be encoded as L3 interface
OutInterface	Egress Logical Interface (LIF/RIF), should be encoded as L3 interface
OutVportClass	OutLIF Profile, by Gport
PortClassPacketProcessing	PP port profile
PortClassTrafficManagement	TM port profile
OutPortTrafficManagement	Out TM-port
InVPort0	In LIF 0, value should be GPORT with subtype LIF
OutVPort0	OutLIF 0, value should be GPORT with subtype LIF
OutVPort1	OutLIF 1, value should be GPORT with subtype LIF
OutVPort2	OutLIF 2, value should be GPORT with subtype LIF
OutVPort3	OutLIF 3, value should be GPORT with subtype LIF
ForwardingAdditionalInfo	Qualifies on the Forwarding additional info
InVportClass0	InLIF Profile 0, by Gport
InVPort0Raw	Qualifies on In LIF 0 input expects raw value
OutVPort0Raw	Qualifies on OutLIF 0 input expects raw value
OutVPort1Raw	Qualifies on OutLIF 1 input expects raw value
OutVPort2Raw	Qualifies on OutLIF 2 input expects raw value
OutVPort3Raw	Qualifies on OutLIF 3 input expects raw value
LearnVsi	Qualifies on learn VSI
LearnData	Qualifies on data field in the Learn Info
VrfValue	Qualifies on the VRF part of forwarding data, without checking that the type is VRF
DstSysPortExtPresent	Qualifies upon if TM Destination extension header is present
SrcPortRaw	Qualifies on source port, input expects raw value
ColorRaw	Qualifies on packet color, input expects raw value
DstPortRaw	Qualifies on destination port, input expects raw value

Table 64: List of Egress-PMF Qualifiers (Continued)

bcmFieldQualify…	Description
OutPortRaw	Qualifies on Egress port. Does not include the core ID, caution should be taken when not adding entry by core, input expects raw value
McastPkt	All ingress replicas will have this value set to 1
LearnExtensionPresent	Qualifies on whether PPH learn extension is present
FtmhAsePresent	Qualify packet on FTMH ASE present field

NOTE: Qualifiers sizes can be obtained by using the following shell command:

```
field map qualifier bcm <stage=ipmf1/2/3..epmf/>
```

11.34.3 Actions List

To see latest version support, run the following diagnostic command:

```
field action Predefined stage=iPMF1/2/3/ePMF
```

Table 65: iPMF-1,2 Actions

bcmFieldAction…	Description
PrioIntNew	TC update
EcnNew	Change metadata ECN / congestion information (including CNI)
Redirect	Redirect packet to to destination
RedirectMcast	Redirect to Multicast Group Id
MirrorIngress	set mirror, value should be encoded as GPORT Type Mirror for mirror command
L3Switch	Sets packet destination, input expects L3 interface
DropPrecedence	Set Packet's Drop Precedence, the values are BCM_FIELD_COLOR_XXX
VrfSet	Set VRF. value[0]: fwd_domain_id_type. value[1]: fwd_domain_id_value. value[1]: fwd_domain_profile. In iPMF3, fwd_domain_profile isn't updated
InnerVlanNew	Replace inner VLAN Id
OuterVlanNew	Replace outer VLAN Id
OuterVlanPrioNew	Replace outer VLAN tag priority
VSQ	Assign matching packets to specified VSQ
Trap	Set Trap code, use TRAP GPORT type value[0]=TRAP_GPORT value[1]=Trap Qualifier. at egress stage refers only to the TRAP ID (strength is not relevant, and can be changed using bcmFieldActionTrapStrength action)
UsePolicerResult	Specify/override where policer result will be used for matched packets
Snoop	Snoop matched packets, input can be either a snoop command: value[0] = Mirror Gport, value[1]=snoop qualifier, value[2]=snoop strength OR: snoop code: value[0] = Trap Gport, value[1]=snoop qualifier, at egress stage refers only to the snoop ID (strength is not relevant, and can be changed using bcmFieldActionSnoopStrength action)
TtlSet	New TTL
DstRpfGportNew	Set the RPF Destination
SrcGportNew	Set the Source-Port
SystemHeaderSet	Modify EEI
VSwitchNew	New VSI. value[0]: fwd_domain_id_type. value[1]: fwd_domain_id_value. value[1]: fwd_domain_profile. In iPMF3, fwd_domain_profile isn't updated
VlanActionSetNew	Modify the VLAN Action Set Id

Table 65: iPMF-1,2 Actions (Continued)

bcmFieldAction…	Description
IngressGportSet	Set a new ingress Global Lif encoded, Values should be Encoded as GPORT
StackingRouteNew	Replace the value of the stacking route
PphPresentSet	If set, a packet processing header is present
FabricHeaderSet	Change System Header Profile, bcm_field_system_header_profile_t
Oam	Changes 4 OAM signals. value[0]: OAM-Stamp-Offset. value[1]: OAM-offset. value[2]: OAM-Sub-Type. value[3]: OAM-Up-Mep
PacketTraceEnable	Enable Trace packet
IEEE1588	Setting various parameters for 1588 frames. Bits(0:7): header_offset. Bit(8): encapsulation. Bits(9:10) command. Bit(11): compensate_time_stamp. Bit(12): packet_is_ieee1588
Forward	Forward Destination Raw value, setting to 1's will drop the packet
PphSnoopCode	Set the PPH Snoop code value (Reserved Bits)
IngressTimeStampInsert	Insert IPIPE time stamp
AdmitProfile	Admit Profile
LatencyFlowId	Change the latency flow-Id. value[0] valid. value[1]: latency_flow_id. value[2]: latency_flow_profile
VisibilityEnable	Enables visibility for current packet
InVportClass0	Updates the profile of an in LIF encoded as Gport
InVportClass1	Updates the profile of an in LIF encoded as Gport
NetworkQos	Updates Network Qos value
Void	Void action, used for cascading, saving buffer in the action payload (for FEM use)
Container	This action will be used as container in IPMF2, to parse the action buffer, when performing cascading between IPMF2 and IPMF3
EgressForwardingIndex	Indicate for egress parser which layer is the Forwarding layer
UDHData0	Set the Payload for User-defined Header 0
UDHData1	Set the Payload for User-defined Header 1
UDHData2	Set the Payload for User-defined Header 2
UDHData3	Set the Payload for User-defined Header 3
UDHBase0	Set the Base for User-defined Header 0
UDHBase1	Set the Base for User-defined Header 1
UDHBase2	Set the Base for User-defined Header 2
UDHBase3	Set the Base for User-defined Header 3
IPTProfile	IPT Profile action (used for instrumentation)
StatId0	Updates statistics ID 0
StatId1	Updates statistics ID 1
StatId2	Updates statistics ID 2
StatId3	Updates statistics ID 3
StatId4	Updates statistics ID 4
StatId5	Updates statistics ID 5
StatId6	Updates statistics ID 6
StatId7	Updates statistics ID 7
StatId8	Updates statistics ID 8
StatId9	Updates statistics ID 9
StatProfile0	Updates statistics profile 0. value[0]: is_meter. value[1]: is_lm. value[2]: type. value[3]: valid
StatProfile1	Updates statistics profile 1. value[0]: is_meter. value[1]: is_lm. value[2]: type. value[3]: valid

Table 65: iPMF-1,2 Actions (Continued)

bcmFieldAction…	Description
StatProfile2	Updates statistics profile 2. value[0]: is_meter. value[1]: is_lm. value[2]: type. value[3]: valid
StatProfile3	Updates statistics profile 3. value[0]: is_meter. value[1]: is_lm. value[2]: type. value[3]: valid
StatProfile4	Updates statistics profile 4. value[0]: is_meter. value[1]: is_lm. value[2]: type. value[3]: valid
StatProfile5	Updates statistics profile 5. value[0]: is_meter. value[1]: is_lm. value[2]: type. value[3]: valid
StatProfile6	Updates statistics profile 6. value[0]: is_meter. value[1]: is_lm. value[2]: type. value[3]: valid
StatProfile7	Updates statistics profile 7. value[0]: is_meter. value[1]: is_lm. value[2]: type. value[3]: valid
StatProfile8	Updates statistics profile 8. value[0]: is_meter. value[1]: is_lm. value[2]: type. value[3]: valid
StatProfile9	Updates statistics profile 9. value[0]: is_meter. value[1]: is_lm. value[2]: type. value[3]: valid
InvalidNext	Invalid next Action macro Id
IngressDoNotLearn	Disable Ingress learning
EgressDoNotLearn	Disable Egress learning
VisibilityClear	Disables visibility for current packet
StatMetaData	Set statistics metaData
InVport0	Change the InLIF 0, value should be encoded as a GPORT with LIF subtype
InVport1	Change the InLIF 1, value should be encoded as a GPORT with LIF subtype
OutVport0	Change OutLIF 0, value should be encoded as a GPORT with LIF subtype
OutVport1	Change OutLIF 1, value should be encoded as a GPORT with LIF subtype
OutVport2	Change OutLIF 2, value should be encoded as a GPORT with LIF subtype
OutVport3	Change OutLIF 3, value should be encoded as a GPORT with LIF subtype
ForwardingTypeMeterMap	Maps L2 forwarding type to one of the general meters
StatSampling	Update statistical sampling value, value encoded as Mirror Gport
StatOamLM	Update the counter of OAM LM, the index of the values are bcmFieldStatOamLmIndexXXX
ForwardingLayerIndex	Update forwarding layer index
InInterface0	Change the InLIF 0, value should be encoded as an interface
InInterface1	Change the InLIF 1, value should be encoded as an interface
OutInterface0	Change OutLIF 0, value should be encoded as type interface
OutInterface1	Change OutLIF 1, value should be encoded as type interface
OutInterface2	Change OutLIF 2, value should be encoded as type interface
OutInterface3	Change OutLIF 3, value should be encoded as type interface
BierStringOffset	Updates Bier string offset
BierStringSize	Updates Bier string size
PacketIsBier	Updates is Bier packet
Eventor	Enable push data to eventor
ForwardingAdditionalInfo	Updates forwarding additional information
SmallExemLearn	Updates the Small Exact Match learn info (IPMF2 only)
InVport0Raw	Change the InLIF 0, with raw value
InVport1Raw	Change the InLIF 1, with raw value
OutVport0Raw	Change the OutLIF 0, with raw value
OutVport1Raw	Change the OutLIF 1, with raw value
OutVport2Raw	Change the OutLIF 2, with raw value
OutVport3Raw	Change the OutLIF 3, with raw value
IngressTimeStampInsertValid	Insert ingress timestamp. Overrides the time stamp in TSH header

Table 65: iPMF-1,2 Actions (Continued)

bcmFieldAction…	Description
SnoopRaw	Snoop matched packets, input should be raw value. At egress stage refers only to the snoop ID (strength is not relevant, and can be changed using bcmFieldActionSnoopStrength action)
DropPrecedenceRaw	Set Packet's Drop Precedence, input expects raw value
FabricHeaderSetRaw	Change System Header Profile, input expects raw value
SrcGportNewRaw	Set the Source-Port, input expects raw value
LatencyFlowIdRaw	Change the latency flow-Id, input expects raw value
MirrorIngressRaw	Set mirror in ingress, input expects raw value
OamRaw	Changes 4 OAM signals (OAM-Stamp-Offset, OAM-offset, OAM-Sub-Type, OAM-Up-Mep), input expects raw value
TrapRaw	Set Trap code, input should be raw value. At egress stage refers only to the TRAP ID (strength is not relevant, and can be changed using bcmFieldActionTrapStrength action)
VrfSetRaw	Set VRF, input expects raw value
StatProfile0Raw	Updates statistics profile 0. Input expects raw value
StatProfile1Raw	Updates statistics profile 1. Input expects raw value
StatProfile2Raw	Updates statistics profile 2. Input expects raw value
StatProfile3Raw	Updates statistics profile 3. Input expects raw value
StatProfile4Raw	Updates statistics profile 4. Input expects raw value
StatProfile5Raw	Updates statistics profile 5. Input expects raw value
StatProfile6Raw	Updates statistics profile 6. Input expects raw value
StatProfile7Raw	Updates statistics profile 7. Input expects raw value
StatProfile8Raw	Updates statistics profile 8. Input expects raw value
StatProfile9Raw	Updates statistics profile 9. Input expects raw value
StatSamplingRaw	Update statistical sampling value, input expects raw value
IpMulticastCompatible	Designates when a Compatible MC procedure found a match
RpfOutVport	Updates the RPF OUT LIF. value[0]: route_valid, value[1]: default_route_found, value[2]: rpf_out_lif (encoded as GPORT with LIF subtype)
RpfOutInterface	Updates the RPF OUT LIF. value[0]: route_valid, value[1]: default_route_found, value[2]: rpf_out_lif (encoded as L3 interface)
RpfOutVportRaw	Updates the RPF OUT LIF. Input expects raw value
LearnRaw0	Updates learn info 0
LearnRaw1	Updates learn info 1
LearnRaw2	Updates learn info 2
LearnRaw3	Updates learn info 3
LearnRaw4	Updates learn info 4
DstRpfGportNewValid	Set the RPF Destination Valid
DstRpfGportNewRaw	Set the RPF Destination. Input expects raw value
StatOamLMRaw	Update the counter of OAM LM with a raw value
LearnKey0	Updates bits 31:0 of learn info key
LearnKey1	Updates bits 63:32 of learn info key
LearnKey2	Updates learn info key and app_db_index. value[0]: bits 79:64 of learn info ley, value[1] app_db_index
LearnKey2Raw	Updates learn info key and app_db_index. Key is at the LSB
LearnPayload0	Updates bits 31:0 of learn info data (payload)
LearnPayload1	Updates bits 63:32 of learn info data (payload)

Table 65: iPMF-1,2 Actions (Continued)

bcmFieldAction…	Description
LearnEntryFormatIndex	Updates entry format index in learn info
LearnOrTransplant	Updates learn or transplant bit
NetworkLoadBalanceKey	Set the network load balancing Key

Table 66: iPMF-3 Actions

bcmFieldAction…	Description
PrioIntNew	TC update
EcnNew	Change metadata ECN / congestion information (including CNI)
Redirect	Redirect packet to to destination
RedirectMcast	Redirect to Multicast Group Id
MirrorIngress	set mirror, value should be encoded as GPORT Type Mirror for mirror command
L3Switch	Sets packet destination, input expects L3 interface
DropPrecedence	Set Packet's Drop Precedence, the values are BCM_FIELD_COLOR_XXX
VrfSet	Set VRF. value[0]: fwd_domain_id_type. value[1]: fwd_domain_id_value. value[1]: fwd_domain_profile. In iPMF3, fwd_domain_profile isn't updated
InnerVlanNew	Replace inner VLAN Id
OuterVlanNew	Replace outer VLAN Id
OuterVlanPrioNew	Replace outer VLAN tag priority
VSQ	Assign matching packets to specified VSQ
Trap	Set Trap code, use TRAP GPORT type value[0]=TRAP_GPORT value[1]=Trap Qualifier. at egress stage refers only to the TRAP ID (strength is not relevant, and can be changed using bcmFieldActionTrapStrength action)
UsePolicerResult	Specify/override where policer result will be used for matched packets
Snoop	Snoop matched packets, input can be either a snoop command: value[0] = Mirror Gport, value[1]=snoop qualifier, value[2]=snoop strength OR: snoop code: value[0] = Trap Gport, value[1]=snoop qualifier, at egress stage refers only to the snoop ID (strength is not relevant, and can be changed using bcmFieldActionSnoopStrength action)
TtlSet	New TTL
TrunkHashKeySet	Set the Trunk Hash Key
SrcGportNew	Set the Source-Port
StartPacketStrip	Strip the start of packet. value[0]: bcm_field_start_packet_strip_t value[1]: extra bytes to remove
SystemHeaderSet	Modify EEI
VSwitchNew	New VSI. value[0]: fwd_domain_id_type. value[1]: fwd_domain_id_value. value[1]: fwd_domain_profile. In iPMF3, fwd_domain_profile isn't updated
VlanActionSetNew	Modify the VLAN Action Set Id
IngressGportSet	Set a new ingress Global LIF encoded, Values should be Encoded as GPORT
StackingRouteNew	Replace the value of the stacking route
PphPresentSet	If set, a packet processing header is present
FabricHeaderSet	Change System Header Profile, bcm_field_system_header_profile_t
Oam	Changes 4 OAM signals. value[0]: OAM-Stamp-Offset. value[1]: OAM-offset. value[2]: OAM-Sub-Type. value[3]: OAM-Up-Mep
IEEE1588	Setting various parameters for 1588 frames. Bits(0:7): header_offset. Bit(8): encapsulation. Bits(9:10) command. Bit(11): compensate_time_stamp. Bit(12): packet_is_ieee1588
Forward	Forward Destination Raw value, setting to 1's will drop the packet

Table 66: iPMF-3 Actions (Continued)

bcmFieldAction…	Description
PphSnoopCode	Set the PPH Snoop code value (Reserved Bits)
AdmitProfile	Admit Profile
LatencyFlowId	Change the latency flow-Id. value[0] valid. value[1]: latency_flow_id. value[2]: latency_flow_profile
VisibilityEnable	Enables visibility for current packet
SystemHeaderSizeAdjust	System Header Size adjust
InVportClass0	Updates the profile of an in LIF encoded as Gport
InVportClass1	Updates the profile of an in LIF encoded as Gport
NetworkQos	Updates Network Qos value
Void	Void action, used for cascading, saving buffer in the action payload (for FEM use)
ParsingStartType	Indicate the layer type that egress parser should consider. For information about the possible layer types, see Section 11.30, Common Types.
ParsingStartOffset	Indicates from which offset egress parser should start parsing (all info before that will not be parsed)
EgressForwardingIndex	Indicate for egress parser which layer is the Forwarding layer
UDHData0	Set the Payload for User-defined Header 0
UDHData1	Set the Payload for User-defined Header 1
UDHData2	Set the Payload for User-defined Header 2
UDHData3	Set the Payload for User-defined Header 3
UDHBase	Set the Base for All User-defined Header 0-3, UDHBase0 is at LSB
IPTProfile	IPT Profile action (used for instrumentation)
IPTCommand	IPT Profile command. value[0]: int_commnad. value[1]: int_profile
StatId0	Updates statistics ID 0
StatId1	Updates statistics ID 1
StatId2	Updates statistics ID 2
StatId3	Updates statistics ID 3
StatId4	Updates statistics ID 4
StatId5	Updates statistics ID 5
StatId6	Updates statistics ID 6
StatId7	Updates statistics ID 7
StatId8	Updates statistics ID 8
StatId9	Updates statistics ID 9
StatProfile0	Updates statistics profile 0. value[0]: is_meter. value[1]: is_lm. value[2]: type. value[3]: valid
StatProfile1	Updates statistics profile 1. value[0]: is_meter. value[1]: is_lm. value[2]: type. value[3]: valid
StatProfile2	Updates statistics profile 2. value[0]: is_meter. value[1]: is_lm. value[2]: type. value[3]: valid
StatProfile3	Updates statistics profile 3. value[0]: is_meter. value[1]: is_lm. value[2]: type. value[3]: valid
StatProfile4	Updates statistics profile 4. value[0]: is_meter. value[1]: is_lm. value[2]: type. value[3]: valid
StatProfile5	Updates statistics profile 5. value[0]: is_meter. value[1]: is_lm. value[2]: type. value[3]: valid
StatProfile6	Updates statistics profile 6. value[0]: is_meter. value[1]: is_lm. value[2]: type. value[3]: valid
StatProfile7	Updates statistics profile 7. value[0]: is_meter. value[1]: is_lm. value[2]: type. value[3]: valid
StatProfile8	Updates statistics profile 8. value[0]: is_meter. value[1]: is_lm. value[2]: type. value[3]: valid
StatProfile9	Updates statistics profile 9. value[0]: is_meter. value[1]: is_lm. value[2]: type. value[3]: valid
InvalidNext	Invalid next Action macro Id

Table 66: iPMF-3 Actions (Continued)

bcmFieldAction…	Description
StatMetaData	Set statistics metaData
InVport0	Change the InLIF 0, value should be encoded as a GPORT with LIF subtype
InVport1	Change the InLIF 1, value should be encoded as a GPORT with LIF subtype
OutVport0	Change OutLIF 0, value should be encoded as a GPORT with LIF subtype
OutVport1	Change OutLIF 1, value should be encoded as a GPORT with LIF subtype
OutVport2	Change OutLIF 2, value should be encoded as a GPORT with LIF subtype
OutVport3	Change OutLIF 3, value should be encoded as a GPORT with LIF subtype
ForwardingTypeMeterMap	Maps L2 forwarding type to one of the general meters
StatSampling	Update statistical sampling value, value encoded as Mirror Gport
StatOamLM	Update the counter of OAM LM, the index of the values are bcmFieldStatOamLmIndexXXX
ForwardingLayerIndex	Update forwarding layer index
InInterface0	Change the InLIF 0, value should be encoded as an interface
InInterface1	Change the InLIF 1, value should be encoded as an interface
OutInterface0	Change OutLIF 0, value should be encoded as type interface
OutInterface1	Change OutLIF 1, value should be encoded as type interface
OutInterface2	Change OutLIF 2, value should be encoded as type interface
OutInterface3	Change OutLIF 3, value should be encoded as type interface
BierStringOffset	Updates Bier string offset
BierStringSize	Updates Bier string size
PacketIsBier	Updates is Bier packet
ForwardingAdditionalInfo	Updates forwarding additional information
InVport0Raw	Change the InLIF 0, with raw value
InVport1Raw	Change the InLIF 1, with raw value
OutVport0Raw	Change the OutLIF 0, with raw value
OutVport1Raw	Change the OutLIF 1, with raw value
OutVport2Raw	Change the OutLIF 2, with raw value
OutVport3Raw	Change the OutLIF 3, with raw value
AppendPointerCompensation	Configures pointer to header append compensation value
IngressTimeStampInsertValid	Insert ingress timestamp. Overrides the time stamp in TSH header
SnoopRaw	Snoop matched packets, input should be raw value. At egress stage refers only to the snoop ID (strength is not relevant, and can be changed using bcmFieldActionSnoopStrength action)
DropPrecedenceRaw	Set Packet's Drop Precedence, input expects raw value
FabricHeaderSetRaw	Change System Header Profile, input expects raw value
SrcGportNewRaw	Set the Source-Port, input expects raw value
LatencyFlowIdRaw	Change the latency flow-Id, input expects raw value
MirrorIngressRaw	Set mirror in ingress, input expects raw value
OamRaw	Changes 4 OAM signals (OAM-Stamp-Offset, OAM-offset, OAM-Sub-Type, OAM-Up-Mep), input expects raw value
TrapRaw	Set Trap code, input should be raw value. At egress stage refers only to the TRAP ID (strength is not relevant, and can be changed using bcmFieldActionTrapStrength action)
StatProfile0Raw	Updates statistics profile 0. Input expects raw value
StatProfile1Raw	Updates statistics profile 1. Input expects raw value
StatProfile2Raw	Updates statistics profile 2. Input expects raw value

Table 66: iPMF-3 Actions (Continued)

bcmFieldAction…	Description
StatProfile3Raw	Updates statistics profile 3. Input expects raw value
StatProfile4Raw	Updates statistics profile 4. Input expects raw value
StatProfile5Raw	Updates statistics profile 5. Input expects raw value
StatProfile6Raw	Updates statistics profile 6. Input expects raw value
StatProfile7Raw	Updates statistics profile 7. Input expects raw value
StatProfile8Raw	Updates statistics profile 8. Input expects raw value
StatProfile9Raw	Updates statistics profile 9. Input expects raw value
StatSamplingRaw	Update statistical sampling value, input expects raw value
StartPacketStripRaw	Strip the start of packet. Refers to bcm_field_start_packet_strip_t. Input expects raw value
ParsingStartTypeRaw	Indicate the layer type that egress parser should consider, input expects raw value
ParsingStartOffsetRaw	Indicate from which offset egress parser should start parsing (all info before that will not be parsed). Input expects raw value
IPTCommandRaw	IPT Profile command. Input expects raw value
LearnRaw0	Updates learn info 0
LearnRaw1	Updates learn info 1
LearnRaw2	Updates learn info 2
LearnRaw3	Updates learn info 3
LearnRaw4	Updates learn info 4
StatOamLMRaw	Update the counter of OAM LM with a raw value
LearnKey0	Updates bits 31:0 of learn info key
LearnKey1	Updates bits 63:32 of learn info key
LearnKey2	Updates learn info key and app_db_index. value[0]: bits 79:64 of learn info ley, value[1] app_db_index
LearnKey2Raw	Updates learn info key and app_db_index. Key is at the LSB
LearnPayload0	Updates bits 31:0 of learn info data (payload)
LearnPayload1	Updates bits 63:32 of learn info data (payload)
LearnEntryFormatIndex	Updates entry format index in learn info
LearnOrTransplant	Updates learn or transplant bit
NetworkLoadBalanceKey	Set the network load balancing key

Table 67: Egress-PMF Actions

bcmFieldAction…	Description
PrioIntNew	TC update **NOTE:** In the ePMF, setting MC traffic to TC=4 is not allowed and can cause traffic issues. Do not use this setting in that case.
Drop	Discard packet
MirrorEgress	Update the mirror profile, mirror_cmd should be encoded as MIRROR Gport
DropPrecedence	Set Packet's Drop Precedence, the values are BCM_FIELD_COLOR_XXX
Trap	Set Trap code, use TRAP GPORT type value[0]=TRAP_GPORT value[1]=Trap Qualifier. at egress stage refers only to the TRAP ID (strength is not relevant, and can be changed using bcmFieldActionTrapStrength action)

Table 67: Egress-PMF Actions (Continued)

bcmFieldAction…	Description
Snoop	Snoop matched packets, input can be either a snoop command: value[0] = Mirror Gport, value[1]=snoop qualifier, value[2]=snoop strength OR: snoop code: value[0] = Trap Gport, value[1]=snoop qualifier, at egress stage refers only to the snoop ID (strength is not relevant, and can be changed using bcmFieldActionSnoopStrength action)
QosMapIdNew	Set the QoS profile
Void	Void action, used for cascading, saving buffer in the action payload (for FEM use)
StatId0	Updates statistics ID 0
StatId1	Updates statistics ID 1
StatProfile0	Updates statistics profile 0. value[0]: is_meter. value[1]: is_lm. value[2]: type. value[3]: valid
StatProfile1	Updates statistics profile 1. value[0]: is_meter. value[1]: is_lm. value[2]: type. value[3]: valid
InvalidNext	Invalid next Action macro Id
AceEntryId	Pointer to the ACE entry ID
TrapStrength	Updates the Trap Strength. Use TRAP Gport
SnoopStrength	updates the snoop strength
OutPortTrafficManagement	Updates the Out-TM port
SnoopRaw	Snoop matched packets, input should be raw value. At egress stage refers only to the snoop ID (strength is not relevant, and can be changed using bcmFieldActionSnoopStrength action)
DropPrecedenceRaw	Set Packet's Drop Precedence, input expects raw value
MirrorEgressRaw	Update the egress mirror profile, input expects raw value
TrapStrengthRaw	Updates the Trap Strength, input expects raw value
SnoopStrengthRaw	Updates the Snoop Strength, input expects raw value

Table 68: Egress-extension (ACE) PMF Actions

bcmFieldAction…	Description
PrioIntNew	TC update
EcnNew	Change metadata ECN / congestion information (including CNI)
DropPrecedence	Set Packet's Drop Precedence, the values are BCM_FIELD_COLOR_XXX
Stat0	Stat meter object. value[0]: stat_meter_obj_cmd. value[1]: stat_meter_obj_id
Stat1	Stat counter object. value[0]: stat_counter_obj_cmd. value[1]: stat_counter_obj_id. value[2]: valid
TtlSet	New TTL
LearnSrcPortNew	Replace the learnt packet source Port
InterfaceClassVPort	Set InLIF profile
Void	Void action, used for cascading, saving buffer in the action payload (for FEM use) or External lookup result
IPTProfile	IPT Profile action (used for instrumentation)
InvalidNext	Invalid next Action macro Id
OutVport0	Change OutLIF 0, value should be encoded as a GPORT with LIF subtype
OutVport1	Change OutLIF 1, value should be encoded as a GPORT with LIF subtype
OutVport2	Change OutLIF 2, value should be encoded as a GPORT with LIF subtype
OutVport3	Change OutLIF 3, value should be encoded as a GPORT with LIF subtype
ForwardingLayerIndex	Update forwarding layer index
OutInterface0	Change OutLIF 0, value should be encoded as type interface

Table 68: Egress-extension (ACE) PMF Actions (Continued)

bcmFieldAction…	Description
OutInterface1	Change OutLIF 1, value should be encoded as type interface
OutInterface2	Change OutLIF 2, value should be encoded as type interface
OutInterface3	Change OutLIF 3, value should be encoded as type interface
AceContextValue	Updates programmable value per ACE context, bcm_field_ace_context_t. Up to four different values can be given in value[0-3], although some may be mutually exclusive
ForwardingAdditionalInfo	Updates forwarding additional information
OutVport0Raw	Change the OutLIF 0, with raw value
OutVport1Raw	Change the OutLIF 1, with raw value
OutVport2Raw	Change the OutLIF 2, with raw value
OutVport3Raw	Change the OutLIF 3, with raw value
DropPrecedenceRaw	Set Packet's Drop Precedence, input expects raw value
AceContextValueRaw	Updates programmable value per ACE context, input expects raw value
Stat0Raw	Stat meter object. Input expects raw value
Stat1Raw	Stat counter object. Input expects raw value

NOTE: Action sizes can be obtained by using the following shell command:

```
field map action bcm <stage=ipmf1/2/3..epmf>
```

Chapter 12: Traps

12.1 Introduction

A trap is a filter that occurs in the device. When a packet travels through the pipeline, it is checked for certain trap conditions. The user can configure an action to be applied to the packet when the trap conditions are met. This means when the packet reaches the trap resolution stage, a configurable action is performed on the related packet. This configurable action might be to drop the packet, redirect the packet to the CPU, redirect the packet to some port in the system, change the priority, or some other action.

The trap mechanism allows the user to change the forward action based on certain conditions. Traps can occur in different blocks of the device, ingress, egress receive packet processor (ERPP) and egress transmit packet processor (ETPP), and also in different stages such as forwarding and packet termination.

The trap mechanism may be used to assign a forward action profile or a snoop action. When referring to the forward action profile, it may be called Forward-Action-Profile or Trap-Code. It is important to differentiate between Forward-Action and Forward-Action-Profile because they are two different entities. The Forward-Action is assigned as a result of forwarding classification, while the Forward-Action-Profile is assigned by the trap mechanism.

12.1.1 Working with Traps

Trap APIs allow the user to configure the Trap-Action-Profile for different trap conditions. This includes configuring the trap code, forward strength, and snoop strength.

12.1.1.1 Trap Code

A Trap-Action-Profile is mapped to a snoop action or a forward action.

12.1.1.2 Forward and Snoop Strength

The forward and snoop strengths are used to resolve a conflict between different filters that attempt to assign different profiles to the same action (Forward/Snoop). Actions assigned the trap with the higher strength.

12.1.1.3 Action Resolution

A trap code and forward strength are stored in the metadata of the packet as it traverses the PP pipe. Every time a trap occurs, a strength check is performed.This check is done between the trap forward strength and the current forward strength that already exists in the metadata of the packet. If the trap forward strength is higher than the existing forward strength value, the metadata is updated with the new trap code and forward strength.

At the end of the pipe, an action resolution is performed. Only a trap code that reaches the action resolution stage is used to update the packet's actions.

A similar process occurs for the snoop strength, which is also traversing with the packet.

12.1.1.4 Action Update

The Forward-Action-Profile may override some or all of the Forward-Action attributes (Destination, TC, DP...). Attributes of the Forward-Action-Profile with an overwrite indication will be updated into the Forwarding-Action.

NOTE: There are two options to drop a packet in the PP:

- Sending the packet to an undefined destination (*black hole*)
- Configuring a trap with a drop destination

The second option is preferred because traps provide more options to monitor those drops. For example, every trap can be associated with a counter.

12.1.2 Trigger Events

There are several mechanisms for trapping a packet. The following table lists the different mechanisms that can trigger traps.

Trigger Event	Location	Purpose
Hard coded	Predefined packet types Header contents	Trap packet of specific type and change its forward action
Protocol	Packet headers, such as L2CP	Per LIF/In-Port for specific protocols
Programmable	Forwarding header content	Create custom trap on forwarding header
Packet checks	—	Packet checks per LIF/RIF (for example: MTU filtering)
Forwarding lookups	—	Destination encoding trap code and strength
OAM	OAM packets	Recycle and redirect packets to a desired destination
LIF traps	—	Change forwarding action of packets coming from a specific LIF/RIF

12.2 Trap Types

Traps can be categorized as either ingress or egress traps:

- Ingress Traps
  - Ingress predefined traps, such as:
    - TTL0 for IPv4 packet (bcmRxTrapIForwardingIpv4Ttl0)
    - MAC SA address equals MAC DA address in (bcmRxTrapLinkLayerSaEqualsDa)
  - Ingress user-defined traps – bcmRxTrapUserDefine
- Egress Traps
  - ERPP user-defined traps – bcmRxTrapEgUserDefine
  - ETPP OAM traps – Such as bcmRxTrapEgTxOamPassive
  - ETPP user-defined traps – bcmRxTrapEgTxUserDefine

Additionally, trap types are used for action configuration or application configuration:

- Action configuration – Create and set the action profile of a specific trap type:
  - Ingress predefined traps – bcmRxTrapForwardingIpv4Ttl0
  - Ingress user-defined traps – bcmRxTrapUserDefine
  - ERPP user-defined traps – bcmRxTrapEgUserDefine
  - ETPP OAM traps – Such as bcmRxTrapEgTxOamPassive
  - ETPP user-defined traps – bcmRxTrapEgTxUserDefine

- Application configuration – Configure Trap-Action-Profile (Trap-Code, forward strength, and snoop strength) of a specific application trap type:
  - Protocol traps – Indicates protocol type to configure, such as bcmRxTrapArp
  - ERPP application traps – Such as bcmRxTrapEgIpv4Ttl0
  - ETPP application traps – Such as bcmRxTrapEgTxSplitHorizon

12.2.1 Ingress Traps

Ingress Trap-Code (Action profile) is the hardware entry of the configured Action-Profile. Different ingress trap types use different ranges of Action-Profiles.

- Predefined traps – Each predefined trap type has a dedicated Action-Profile.
- User-defined traps.

12.2.1.1 Ingress Predefined Traps

Ingress predefined traps are traps with predefined Trap-Codes that are dedicated hardware entries for the trap action. When trap conditions are answered, the mapping to Forward Action and Snoop Action is predefined by hardware. Each of these traps has Forward strength and Snoop strength registers.

The trap type is defined by the conditions of the packet to be trapped. Trap type (`bcm_rx_trap_t`) is of the form **bcmRxTrapType**, where Type = filter condition. When a trap can occur in several stages, the Type = stage is appended by filter condition, i.e. bcmRxTrapTerminatedIpv4Ttl0 and bcmRxTrapForwardingIpv4Ttl0.

Supported Predefined Trap Types

A list of ingress predefined trap types per device can be found in Section 12.7, Device-Specific Traps.

12.2.1.2 Ingress User-Defined Traps

Ingress user-defined traps are traps with action entries that have no predefined mappings. The user can allocate an available Trap-Code from a pool or allocate a specific Trap-Code if allocation is done using the ID.

No strength configuration is done, since a trap is used by different applications.

Configuration is done by using trap type bcmRxTrapUserDefine.

The following applications use ingress user-defined traps:

- PMF
- FEC
- Protocol traps
- Programmable traps
- Port default behavior
- LIF default behavior

12.2.2 Egress Traps

The three egress trap types are:

- ERPP user-defined traps
- ETPP user-defined traps
- ETPP OAM traps

12.2.2.1 ERPP User-Defined Traps

ERPP user-defined traps are used for allocating an Action-Profile and setting the desired action. Eight Action-Profiles are available for configuration of actions used by ERPP application traps. One profile is reserved as the default profile, and seven profiles are configurable.

- Default profile (saved) – BCM_RX_TRAP_EG_TRAP_ID_DEFAULT.
- Seven configurable profiles – Allocate a profile and set the desired action for the trap. Configuration is done by using trap type bcmRxTrapEgUserDefine.

12.2.2.2 ETPP User-Defined Traps

ETPP user-defined traps are used for allocating an Action-Profile and setting the desired action. Six Action-Profiles are available for configuration of actions used by ETPP application traps.

- Default profile – BCM_RX_TRAP_EG_TX_TRAP_ID_DEFAULT
- Drop packet action profile – BCM_RX_TRAP_EG_TX_TRAP_ID_DROP
- Four user-configurable profiles. Allocate a profile and set the desired action for the trap. Configuration is done by using trap type bcmRxTrapEgTxUserDefine.

An additional dedicated Action-Profile is reserved for visibility traps and is activated when visibility bit equals one. Configuration is done by using trap type bcmRxTrapEgTxVisibility.

12.2.2.3 ETPP OAM Traps

ETPP OAM traps have a predefined Action-Profile.

Trap APIs set the Action-Profile. They do not configure the strengths.

Supported ETPP OAM trap types include:

- bcmRxTrapEgTxOamLevel
- bcmRxTrapEgTxOamPassive
- bcmRxTrapEgTxOamReflector
- bcmRxTrapEgTxOamUpMEPDest1
- bcmRxTrapEgTxOamUpMEPDest2
- bcmRxTrapEgTxOamUpMEPOamp
- bcmRxTrapEgTxIfaEgressMetadata

12.2.3 SOC Properties

N/A

12.2.4 Shell Commands

The following shell commands are available:

- `trap list pre block=<IRPP/ETPP>`
  Shows a list of all configured predefined traps. The information shown for each trap includes:
  - Name
  - trap-id (hardware code)
  - Snoop strength
  - Forward strength
- `trap list usr block=<IRPP/ERPP/ETPP>`
  Shows a list of all allocated user-defined traps and their `trap_id`.
- `trap act info id=<trap_id>`
  Shows the action configured for a specific trap.
- `trap last info core=<core_id> block=<IRPP/ERPP/ETPP>`
  Shows the last packet trap information:
  - Resolved trap for forward/snoop
  - Resolved trap forward/snoop action
  - Considered traps for packet
- `trap map pre`
  - List of all available traps with a short description and their `trap_id`.

12.2.5 Application Reference

Examples of trap basic configuration usage with ingress predefined traps:

- **Type:** CINT reference
- **Path:** `cint_rx_trap_ingress_traps.c`

Examples of trap basic configuration usage with ERPP user-defined traps:

- **Type:** CINT reference
- **Path:** `cint_rx_trap_erpp_traps.c`

Examples of trap basic configuration usage with ETPP user-defined traps:

- **Type:** CINT reference
- **Path:** `cint_rx_trap_etpp_traps.c`

Example of ETPP trap configuration usage with `BCM_RX_TRAP_FLAGS2_KEEP_FABRIC_HEADER`

- **Type:** CINT reference
- **Path:** `cint_rx_trap_etpp_keep_orig_fabric_hdr.c`

Examples of ETPP trap configuration usage with packet recycling and 2nd pass FHEI stamping and updating destination

- **Type:** CINT reference
- **Path:** `cint_rx_trap_etpp_recycle_and_2nd_pass_stamp.c`

12.3 Trap Configuration

Traps have two types of configurations:

- **Action Configurations**—Used to create and set the action profile of a specific trap type.
- **Application Configurations**—Used to configure Trap-Action-Profile (Trap-Code, Fwd Strength, and snoop strength) of a specific application trap type.

12.3.1 Action Configuration

All traps share the same sequence structure. Each trap type has a different set of actions, configured per trap type.

12.3.1.1 Configuration Flow

This section describes the following getters, setters, and a destroyer:

- Forward Action Setter
- Snoop Action Setter
- Forward Action Getter
- Snoop Action Getter
- Destroyer

Forward Action Setter

This action includes creating the trap and setting its configuration.

No.	API	Description
1	bcm_rx_trap_type_create(unit, flags, trap_type, &trap_id);	Allocate hardware resource for action.
2	bcm_rx_trap_config_t_init(&trap_config);	Before setting the desired action, initialize trap configuration struct.
3	bcm_rx_trap_config_t trap_config;	Configure `trap_config.trap_strength` to a value larger than 0 for predefined traps. Configure actions, such as destination, TC, and DP.
4	bcm_rx_trap_set(unit, trap_id, &trap_config);	Set action of trap in hardware.

Snoop Action Setter

This action describes setting the snoop action.

No.	API	Description
1	bcm_rx_trap_type_create(unit, flags, trap_type, &trap_id);	Allocate hardware resource for action.
2	bcm_mirror_destination_create(unit, &mirror_dest);	Allocate hardware resource and configure the snoop action.
3	bcm_rx_trap_config_t_init(&trap_config);	Before setting the desired action, initialize trap configuration struct.
4	bcm_rx_trap_config_t trap_config;	Configure `trap_config.snoop_strength` to a value larger than 0 for predefined traps. `trap_config.snoop_cmnd` = BCM_GPORT_MIRROR_GET(mirror_dest.mirror_dest_id);
5	bcm_rx_trap_set(unit, trap_id, &trap_config);	Set the action of the trap in hardware.

Forward Action Getter

No.	API	Description
1	bcm_rx_trap_get(unit, trap_id, &trap_config);	Get the action of the trap configured from hardware.

Snoop Action Getter

No.	API	Description
1	bcm_rx_trap_get(unit, trap_id, &trap_config);	Get action of trap configured from hardware.
2	BCM_GPORT_MIRROR_SNOOP_SET(mirror_dest_id, trap_config.snoop_cmnd);	Encode mirror destination as snoop.
3	bcm_mirror_destination_get(unit, mirror_dest_id, &mirror_dest);	Get the configuration of the snoop action.

Destroyer

This action includes destroying and clearing the egress trap configuration.

No.	API	Description
1	bcm_rx_trap_get(unit, trap_id, &trap_config);	Get action of trap configured from hardware. **NOTE:** This stage is required only for clearing ingress traps snoop action.
2	BCM_GPORT_MIRROR_SNOOP_SET (mirror_dest_id, trap_config.snoop_cmnd);	Encode mirror destination as snoop. **NOTE:** This stage is required only for clearing ingress traps snoop action.
3	bcm_mirror_destination_destroy(unit, mirror_dest_id);	Clear configuration of snoop action. **NOTE:** This stage is required only for clearing ingress traps snoop action.
4	bcm_rx_trap_type_destroy(unit, trap_id);	Destroy trap and deallocate hardware resources.

12.3.2 Trap Action Configuration

This section describes additional action configurations per trap.

12.3.2.1 bcm_rx_trap_config_t

This section includes **bcm_rx_trap_config_t** action configuration attributes for:

- Ingress traps
- ERPP traps
- ETPP traps

Ingress traps

Field (and Field Type)	Value	Description
flags (uint32)	BCM_RX_TRAP_UPDATE_DEST	If set, packet destination is updated
	BCM_RX_TRAP_UPDATE_PRIO	If set, the packet priority is updated (Traffic Class)
	BCM_RX_TRAP_UPDATE_COLOR	If set, the packet color is updated (Drop Precedence)
	BCM_RX_TRAP_DEST_MULTICAST	Set together to update the multicast destination Field dest_group carries the updated multicast ID
	BCM_RX_TRAP_UPDATE_DEST	
	BCM_RX_TRAP_TRAP	If set, cpu-trap-code is stamped in PP-Header
	BCM_RX_TRAP_BYPASS_FILTERS	If set, Link Layer filters in Egress are ignored
	BCM_RX_TRAP_LEARN_DISABLE	If set, ingress and egress learning is disabled for this forward action
	BCM_RX_TRAP_UPDATE_ADD_VLAN	If set, VSI value is added to the destination
	BCM_RX_TRAP_UPDATE_FORWARDING_HEADER	If set, the user specifies the forwarding header position
	BCM_RX_TRAP_UPDATE_ENCAP_ID	If set, OutLIFs are updated. The fields encap_id and encap_id2 carry the out_lif gport.
	BCM_RX_TRAP_UPDATE_METER_CMD	If set, the meter command is updated
	BCM_RX_TRAP_UPDATE_MAPPED_STRENGTH	If set, the egress strength the trap is updated
	BCM_RX_TRAP_UPDATE_ECN_VALUE	If set, the Ethernet encapsulation value is updated.
	BCM_RX_TRAP_UPDATE_VSQ	If set, the VSQ pointer is updated
	BCM_RX_TRAP_UPDATE_EGRESS_FWD_INDEX	If set, egress parsing index is updated
flags2 (uint32)	BCM_RX_TRAP_FLAGS2_SNOOP_CODE_CLEAR	If set, Snoop code will be set to 0
	BCM_RX_TRAP_FLAGS2_MIRROR_CODE_CLEAR	If set, Mirror code will be set to 0
	BCM_RX_TRAP_FLAGS2_STAT_SAMPLING_CODE_CLEAR	If set, Stat Sampling code will be set to 0
dest_port (bcm_gport_t)	Destination port	**NOTE:** Trunk gport is not supported

Field (and Field Type)	Value	Description
dest_group (bcm_multicast_t)	Multicast destination (multicast ID)	—
prio (int)	Packet internal priority (Traffic Class)	—
color (bcm_color_t)	Packet color (Drop Precedence)	—
forwarding_header (bcm_rx_trap_forwarding_header_t)	Forwarding header position.	Options: start of packet, Ethernet header, first header after Ethernet, etc
encap_id (bcm_if_t)	Encodes OutLIF 0 to overwrite	encap_id needs to contain gport (bcm_gport_t) although it is of type intf (bcm_if_t)
encap_id2 (bcm_if_t)	Encodes OutLIF 1 to overwrite	encap_id2 needs to contain gport (bcm_gport_t) although it is of type intf (bcm_if_t)
core_config_arr (bcm_rx_trap_core_config_t *)	Destinations information per core	—
core_config_arr->dest_port (bcm_gport_t)	Destination port per core	bcm_rx_trap_core_config_t struct fields
core_config_arr->encap_id (bcm_if_t)	Encodes OutLIF 0 to overwrite per core	
core_config_arr->encap_id2 (bcm_if_t)	Encodes OutLIF 1 to overwrite per core	
core_config_arr_len (int)	Length of config core array	—
meter_cmd (int)	Meter command	—
mapped_trap_strength (uint8)	egress forward strength	Used to overwrite egress strength.
ecn_value (uint8)	Ethernet encapsulation	—
vsq (uint16)	Statistics VSQ pointer	—
latency_flow_id_clear (uint8)	Indication if to clear latency_flow_id	TRUE/FALSE value
visibility_value (uint8)	Visibility value	TRUE/FALSE value
egress_forwarding_index (uint32)	Egress forwarding parsing index	—
stat_clear_bitmap (uint32)	Bitmap indicating which statistical objects to clear	To set all statistical objects to be valid, set stat_clear_bitmap to 0x3FF
stat_obj_config_arr (bcm_rx_trap_stat_obj_config_t * [BCM_RX_TRAP_MAX_STAT_OBJ_ARR_LEN])	Overwrite configuration of statistical objects	—
stat_obj_config_arr[i].counter_command_id (int)	Command id (interface id), 0-9	bcm_rx_trap_stat_obj_config_t struct fields
stat_obj_config_arr[i].stat_id (uint32)	Object statistic id (the counter_pointer value as sent to the CRPS)	
stat_obj_config_arr[i].is_offset_by_qual_enable (uint8)	StatObjValue is added a trap qualifier offset	
stat_obj_config_arr[i].stat_object_type (int)	Statistical object type	
stat_obj_config_arr[i].is_meter (uint8)	Allow metering	
stat_obj_config_len (uint32)	stat_obj_config_arr length	Up to 2 objects
stat_metadata_mask(uint32)	Statistical metadata mask	ResolvedStatisticMetaData = InStatisticMetaData | stat_metadata_mask
snoop_cmnd (int)	Snoop command	—

Field (and Field Type)	Value	Description
snoop_strength (int)	Snoop strength.	Not relevant for strengthless ingress traps. (In other words, user-defined and OAM traps.
trap_strength (int)	Forward strength.	Not relevant for strengthless ingress traps. (In other words, user-defined and OAM traps.

ERPP Traps

Field (and Field Type)	Value	Description
flags (uint32)	BCM_RX_TRAP_UPDATE_DEST	If set, the packet destination is updated
	BCM_RX_TRAP_UPDATE_PRIO	If set, the packet priority is updated (Traffic Class)
	BCM_RX_TRAP_UPDATE_COLOR	If set, the color of the packet is updated (Drop Precedence).
	BCM_RX_TRAP_UPDATE_QOS_MAP_ID	If set, quality of service profile is updated
	BCM_RX_TRAP_UPDATE_ENCAP_ID	If set, CUD is updated
	BCM_RX_TRAP_UPDATE_COUNTER	If set, PMF Counter 0 is enabled.
	BCM_RX_TRAP_UPDATE_COUNTER_2	If set, PMF Counter 1 is enabled.
dest_port (bcm_gport_t)	Destination port	Used for configuring drop (BCM_GPORT_BLACK_HOLE) or recycle port. **NOTE:** Trunk gport is not supported
core_config_arr (bcm_rx_trap_core_config_t *)	Destinations information per core	—
core_config_arr → dest_port (bcm_gport_t)	Destination port per core	bcm_rx_trap_core_config_t struct fields **NOTE:** Trunk gport is not supported
core_config_arr_len (int)	Length of config core array	—
prio (int)	Packet internal priority (Traffic Class)	—
color (bcm_color_t)	Packet color (Drop Precedence)	—
qos_map_id (int)	Quality of service ID	—
encap_id (bcm_if_t)	OutLIF (encoded) or ACE pointer	—
stat_obj_config_arr (bcm_rx_trap_stat_obj_config_t * [BCM_RX_TRAP_MAX_STAT_OBJ_ARR_LEN])	Overwrite configuration of statistical objects. Only index 0 and 1 are valid.	—
stat_obj_config_arr[i].stat_profile (int)	Statistical Object Profile	—
stat_obj_config_arr[i].stat_id (uint32)	Object statistic id (the counter_pointer value)	—
mirror_profile (int)	Mirror Action profile	—
snoop_cmnd (int)	Snoop Profile	—
pp_drop_reason (int)	Packet Processing drop reason	Update 2 bits in counter metadata.

ETPP Traps

Field (and Field Type)	Value	Description
flags (uint32)	BCM_RX_TRAP_COPY_DROP	If set, ETPP trap will drop recycle copy packet (only for user profiles and visibility).
flags2 (uint32)	BCM_RX_TRAP_FLAGS2_KEEP_FABRIC_HEADER	If set, the Recycled packet will have the original fabric headers, that is, system headers and terminated headers of the original packet (termination and encapsulation of ETPP are disabled in this case). **NOTE:** Relevant only for user-defined profiles.
	BCM_RX_TRAP_FLAGS2_INT_STAT_DISABLE	If set, internal statistics are disabled. **NOTE:** Relevant only for drop profile.
	BCM_RX_TRAP_FLAGS2_MIRROR_CODE_CLEAR	If set, the mirror copy is discarded. **NOTE:** This action cannot be configured for OAM ETPP traps.
	BCM_RX_TRAP_FLAGS2_SNOOP_CODE_CLEAR	If set, snoop copy is discarded. **NOTE:** This action cannot be configured for OAM ETPP traps.
cpu_trap_gport (bcm_gport_t)	Encodes the Trap-Code and forward strength of ingress trap executed on recycled packet	—
stamped_trap_code (int)	Trap-Code stamped on FHEI header	Relevant only for ETPP OAM traps
is_recycle_high_priority (uint8)	Recycle high priority indication	TRUE/FALSE value
is_recycle_crop_pkt (uint8)	Crop recycle packet indication	TRUE/FALSE value
is_recycle_append_ftmh (uint8)	Indication regarding appending the original FTMH to the recycled packet	TRUE/FALSE value
pp_drop_reason (int)	Packet processing drop reason	Update 2 bits in counter metadata. **NOTE:** This action can be configured to drop profiles and user-defined profiles.

12.4 Trapping and Redirecting a Packet to the CPU

Packets can be trapped and redirected to the CPU in all stages of the pipeline. Using the trap mechanism, this is done by changing the forwarding destination of the packet to a CPU port. The process for forwarding to the CPU depends on the stage in the pipeline.

The ingress direction has a default trap for redirecting to the CPU (bcmRxTrapDfltRedirectToCpuPacket), which is treated as a user-defined trap with a preset action. It must be supplied to the PMF as an action or to the LIF, protocol, or programmable ingress configurable traps. Predefined ingress traps can also be configured to redirect a packet to the CPU by setting the update destination flag (BCM_RX_TRAP_UPDATE_DEST) and destination port to BCM_GPORT_LOCAL_CPU.

If the packet must be sent to the CPU by ERPP or ETPP traps, it must be recycled first and then have its destination changed by the ingress on the second pass. Based on the stage, the recycling and subsequent second-pass trapping and sending to a CPU port are done differently for ERPP and ETPP.

In the ERPP, the packet must be trapped and sent to a recycle port, which sends it to the ingress again, where it must hit a preconfigured PMF field group or configurable trap as explained previously for redirecting to the CPU in the ingress.

In the ETPP, the packet is recycled automatically if a user-defined trap action profile is hit. To redirect the packet to the CPU, the cpu_trap_gport field indicates, to the ingress, the trap and the strength with which the recycled packet enters the ingress.

12.4.1 Application Reference

Examples of Ingress trap to CPU usage

- **Type**: CINT reference
- **Path**: `cint_rx_trap_icmp_redirect.c`

Examples of ERPP trap to CPU usage

- **Type**: CINT reference
- **Path**: `cint_rx_trap_erpp_trap_to_cpu.c`

Examples of ETPP trap to CPU usage

- **Type**: CINT reference
- **Path**: `cint_rx_trap_etpp_trap_to_cpu.c`

12.5 Application Configuration

This section describes how to configure the Trap-Action-Profile which consists of the Action-Profile, Forward strength and Snoop strength registers.

There are five groups of application traps that require Trap-Action-Profile configuration:

- Egress application Traps
- Protocol Traps
- Programmable Traps
- ETPP MTU Traps
- LIF Traps

12.5.1 Egress Application Traps

After configuring the Action-Profile of an ERPP or ETPP trap, an additional application configuration is required for each egress application trap type.

12.5.1.1 ERPP Application Trap Types

The action-profile for the ERPP application traps is configured using bcmRxTrapEgUserDefine.

For a full list of ERPP application traps per device, see Section 12.7, Device-Specific Traps.

NOTE: bcmRxTrapEgIpv4Ttl0 and bcmRxTrapEgIpv4Ttl1 traps are not restricted to IPv4. They also support IPv6 and MPLS TTL, depending on the forwarding header.

12.5.1.2 ETPP Application Trap Types

The action-profile for the ETPP application traps is configured using bcmRxTrapEgTxUserDefine.

NOTE: Only the bcmRxTrapEgTxLatency trap can configure a mirror-action-profile.

For a full list of ETPP application traps per device, see Section 12.7, Device-Specific Traps.

12.5.1.3 SOC Properties

N/A

12.5.1.4 Configuration Flow

Create trap and set *forward action* configuration flow:

- `bcm_rx_trap_type_create(unit, flags, trap_type, &trap_id);`
  Allocate hardware resource for action, trap_type can be:
  bcmRxTrapEgUserDefine for ERPP
  bcmRxTrapEgTxUserDefine for ETPP
- `bcm_rx_trap_config_t_init(&trap_config);`
  Initialize trap configuration struct *before setting desired action.*
- `bcm_rx_trap_set(unit, trap_id, &trap_config);`
  Set action of trap in hardware
- `BCM_GPORT_TRAP_SET(trap_gport, trap_id, fwd_strength, 0);`
  Encode trap_id, forward strength and snoop strength as gport
- `bcm_rx_trap_action_profile_set(unit, app_flags, app_trap_type, trap_gport);`
  Set the Trap-Action-Profile for trap of specific application trap type

Create trap and set *snoop action* configuration flow:

- `bcm_mirror_destination_create(unit, &mirror_dest);`
  Allocate hardware resource and configure the snoop action
- `snoop_id = BCM_GPORT_MIRROR_GET(mirror_dest.mirror_dest_id);`
  Get the snoop action profile
- `BCM_GPORT_TRAP_SET(trap_gport, snoop_id, 0, snoop_strength);`
  Encode trap_id, forward strength and snoop strength as gport
- `bcm_rx_trap_action_profile_set(unit, app_flags, app_trap_type, trap_gport);`
  Set the Trap-Action-Profile for trap of specific application trap type

Set a mirror action configuration flow:

- `bcm_mirror_destination_create(unit, &mirror_dest);`
  Allocate HW resource and configure the mirror action
- `bcm_rx_trap_action_profile_set(unit,app_flags, bcmRxTrapEgTxLatency, mirror_dest.mirror_dest_id);`
  Set the mirror action profile for bcmRxTrapEgTxLatency trap.

Get *forward action* configuration flow:

- `bcm_rx_trap_action_profile_get(unit, app_trap_type, &trap_gport);`
  Get the Trap-Action-Profile for trap of specific type
- `trap_id = BCM_GPORT_TRAP_GET_ID(trap_gport);`
  Decode trap_id from trap gport
- `bcm_rx_trap_get(unit, trap_id, &trap_config);`
  Get action of trap configured from hardware

Get *snoop action* configuration flow:

- `bcm_rx_trap_action_profile_get(unit, app_trap_type, &trap_gport);`
  Get the Trap-Action-Profile for trap of specific type
- `snoop_id = BCM_GPORT_TRAP_GET_ID(trap_gport);`
  Decode snoop_id from trap gport
- `BCM_GPORT_MIRROR_SNOOP_SET(mirror_dest_id, snoop_id);`
  Encode mirror destination as snoop
- `bcm_mirror_destination_get(unit, mirror_dest_id, &mirror_dest);`
  Get configuration of snoop action

Clear *forward action* configuration flow:

- `bcm_rx_trap_action_profile_get(unit, app_trap_type, &trap_gport);`
  Get the Trap-Action-Profile for trap of specific type
- `bcm_rx_trap_action_profile_clear(unit, app_trap_type);`
  Clear the Trap-Action-Profile for trap of specific type
- `trap_id = BCM_GPORT_TRAP_GET_ID(trap_gport);`
  Decode trap_id from trap gport
- `bcm_rx_trap_type_destroy(unit, trap_id);`
  Destroy trap, deallocate hardware resources

Clear *snoop action* configuration flow:

- `bcm_rx_trap_action_profile_get(unit, app_trap_type, &trap_gport);`
  Get the Trap-Action-Profile for trap of specific type
- `bcm_rx_trap_action_profile_clear(unit, app_trap_type);`
  Clear the Trap-Action-Profile for trap of specific type
- `snoop_id = BCM_GPORT_TRAP_GET_ID(trap_gport);`
  Decode snoop_id from trap gport
- `BCM_GPORT_MIRROR_SNOOP_SET(mirror_dest_id, snoop_id);`
  Encode mirror destination as snoop
- `bcm_mirror_destination_destroy(unit, mirror_dest_id);`
  Clear configuration of snoop action

NOTE: Either a forward action (fwd_strength > 0) *or* a snoop action (snoop_strength > 0) can be configured. Both actions cannot be configured together.

12.5.1.5 Shell Commands

The following shell command is available:

- trap list appl block=<ERPP/ETPP>
  Show a list of all configured application traps. The information shown for each trap includes:
  - name
  - trap_id
  - forward_action_profile
  - snoop strength
  - forward strength
- trap map appl
  - List of all available application traps with a short description.

12.5.1.6 Application Reference

Examples of ERPP application trap usage:

- **Type:** CINT reference
- **Path:** `cint_rx_trap_erpp_traps.c`

Examples of ETPP application trap usage:

- **Type:** CINT reference
- **Path:** `cint_rx_trap_etpp_traps.c`

12.5.2 Protocol Traps

The user can filter packets of certain protocols that are received either from a specific In-PP-Port, from a specific InLIF, or from a specific In-RIF.

A trap is activated by inspecting the headers of packets to detect a certain protocol.

The following table shows the arguments associated with each protocol.

Protocol Type	Arguments
L2CP (aka Reserved Multicast)	6 MAC LSBs (64 options)
ICMP (ICMPv4 and ICMPv6)	ICMP type (256 types)
IGMP	IGMPv1, IGMPv2, IGMPv3
Non-authorized 802.1X	—
DHCP	IPv4-Client, IPv4-Server, IPv6-Client, IPv6-Server
ARP	ARP, My ARP

Protocol traps can be configured per InLIF or per In-PP-port, depending on global device mode initialization (see Section 12.5.2.1, SOC Properties). Each InLIF or In-PP-port configuration contains seven protocol handling profiles (`bcm_rx_trap_protocol_profiles_t`), with one profile per protocol type where the size of each protocol handling profile is 2b.

For each combination of `protocol_type`, argument, and protocol handling profile (`bcm_rx_trap_protocol_key_t`), the user can set a Trap-Action-Profile (Trap-code, Forward strength, and Snoop strength encoded as gport). The Trap-Code indicates the action to be done on the packet and it is allocated by setting a user-defined trap (see Section 12.5.2.2, Configuration Flow).

NOTE: The non-authorized 802.1X handling profile is different from the other protocols:

- 00 (profile 0) – 802.1X authorized state, no trapping.
- 01 (profile 1) – 802.1X non-authorized state, Col-Res=0. Other traps take precedence.
- 1x (profiles 2,3) – 802.1X non-authorized state, Col-Res=1. Take precedence over other traps.

The following table lists the supported trap types (bcm_rx_trap_t).

Trap Type	Details
bcmRxTrapL2cpPeer	L2cp trap
bcmRxTrapIcmpRedirect	ICMP trap
bcmRxTrapIgmpMembershipQuery	IGMP membership query trap (IGMPv1)
bcmRxTrapIgmpReportLeaveMsg	IGMP report leave msg trap (IGMPv2)
bcmRxTrapIgmpUndefined	IGMP undefined trap (IGMPv3)
bcmRxTrap8021xFail	non authorized 802.1x trap
bcmRxTrapDhcpv4Client	DHCP IPv4 client trap
bcmRxTrapDhcpv4Server	DHCP IPv4 server trap
bcmRxTrapDhcpv6Client	DHCP IPv6 client trap
bcmRxTrapDhcpv6Server	DHCP IPv6 server trap
bcmRxTrapArp	ARP trap
bcmRxTrapArpMyIp	MyARP trap

12.5.2.1 SOC Properties

A new SOC property `protocol_traps_mode=IN_LIF/IN_PORT` is added to indicate whether protocol profiles are configured per InLIF or per In-PP-Port.

12.5.2.2 Configuration Flow

The following table describes the structures for the protocol traps.

Struct	Field (and Field Type)	Value
`bcm_rx_trap_protocol_key_t`	`protocol_type` (`bcm_rx_trap_t`)	Protocol trap type
	`protocol_trap_profile` (uint8)	Protocol handling profile (2b)
	`trap_args` (uint8)	Argument according to protocol type L2CP – MAC's 6 LSBs (0 to 63) ICMP – ICMP types (0 to 31)
`bcm_rx_trap_protocol_profiles_t`	`l2cp_trap_profile` (uint8)	L2CP handling profile
	`icmpv4_trap_profile` (uint8)	ICMP IPv4 handling profile
	`icmpv6_trap_profile` (uint8)	ICMP IPv6 handling profile
	`arp_trap_profile` (uint8)	ARP handling profile
	`igmp_trap_profile` (uint8)	IGMP handling profile
	`dhcp_trap_profile` (uint8)	DHCP handling profile
	`non_auth_8021x_trap_profile` (uint8)	Non authorized 802.1x trap profile, MSB – Collision bit LSB – Profile bit

The following tables describe additional configuration for ARP, My ARP, My NDP, and ICMP protocols using `bcm_switch_control_set`, `bcm_switch_control_get`, `bcm_switch_control_indexed_set`, and `bcm_switch_control_indexed_get` APIs.

Protocol	bcmSwitchXXX	Description
ARP	bcmSwitchArpIgnoreDa	0: Ignore DA value 1: Trap only for DA=0xff-ff-ff-ff-ff-ff
My ARP	bcmSwitchArpMyIp1	Configure IP 1 to trap
	bcmSwitchArpMyIp2	Configure IP 2 to trap
My NDP	bcmSwitchNdpMyIp1	Configure IP 1 to trap
	bcmSwitchNdpMyIp2	Configure IP 2 to trap
ICMP	bcmSwitchIcmpIgnoreDa	0: Ignore DA value 1: Trap only for DA=0x33-33-xx-xx-xx-xx

Struct	Field (and Field Type)	Value
`bcm_switch_control_key_t`	type (`bcm_switch_control_t`)	bcmSwitchNdpMyIp1 bcmSwitchNdpMyIp2
	index(int)	bcmSwitchIPv6HighMsb bcmSwitchIPv6HighLsb bcmSwitchIPv6LowMsb bcmSwitchIPv6LowLsb
`bcm_switch_control_info_t`	value (int)	32b word from IPv6 address

The configuration flow is as follows:

1. Set protocol trap configuration flow:
   a. `bcm_rx_trap_type_create(unit, flags, bcmRxTrapUserDefine, &trap_id);`
      Create user-defined trap, allocate hardware resource.
   b. `bcm_rx_trap_set(unit, trap_id, &trap_config);`
      Set action of trap in hardware
   c. `BCM_GPORT_TRAP_SET(trap_gport, trap_id, fwd_strength, snp_strength);`
      Encode trap_id, forward strength and snoop strength as gport.
   d. `bcm_rx_trap_protocol_set(unit, protocol_key, trap_gport);`
      Set the trap action profile for desired protocol and protocol-handling profile.
   e. `bcm_rx_trap_protocol_profiles_set(unit, port, &protocol_profiles);`
      Set protocol profiles for a given LIF/port.

      Or

      `bcm_rx_trap_protocol_interface_profiles_set(unit, intf, &protocol_profiles);`

      Set protocol profiles for the given RIF interface.
2. Perform additional configuration for other protocols:
   a. Additional configuration flow for MyNDP protocol:

      `bcm_switch_control_indexed_set(unit, key, info);`

      Set a 32b word of the NDP MyIP IPv6 address.
   b. Additional configuration flow for MyARP protocol:

      `bcm_switch_control_set(unit, bcmSwitchArpMyIp1/2, my_ip);`

      Set the IPv4 address for ARP MyIP match configuration.
   c. Additional configuration flow for ARP protocol:

      `bcm_switch_control_set(unit, bcmSwitchArpIgnoreDa, TRUE);`

      Enable or disable Ignore DA for Arp protocol.
   d. Additional configuration flow for ICMP protocol:

      `bcm_switch_control_set(unit, bcmSwitchIcmpIgnoreDa , TRUE);`

      Enable or disable Ignore DA for ICMP protocol.

3. Get protocol trap configuration flow:
   a. `bcm_rx_trap_protocol_get(unit, protocol_key, &trap_gport);`
      Get the trap action profile for desired protocol and protocol-handling profile.
   b. `bcm_rx_trap_protocol_profiles_get(unit, port, &protocol_profiles);`
      Get protocol profiles for given LIF/port.

      Or

      `bcm_rx_trap_protocol_interface_profiles_get(unit, intf, &protocol_profiles);`

      Get protocol profiles for the given RIF interface.
4. Perform additional configuration for other protocols:
   a. Additional flow for MyNDP protocol:

      `bcm_switch_control_indexed_get(unit, key, &info);`

      Get a 32b word of the NDP MyIP IPv6 address.
   b. Additional configuration flow for MyARP protocol:

      `bcm_switch_control_get(unit, bcmSwitchArpMyIp1/2, &my_ip);`

      Get the IPv4 address for ARP MyIP match configuration.
   c. Additional configuration flow for ARP protocol:

      `bcm_switch_control_get(unit, bcmSwitchArpIgnoreDa, &value);`

      Enable or disable Ignore DA for Arp protocol.
   d. Additional configuration flow for ICMP protocol:

      `bcm_switch_control_get(unit, bcmSwitchIcmpIgnoreDa , &value);`

      Enable or disable Ignore DA for ICMP protocol.
5. Clear protocol trap sequence:
   a. `bcm_rx_trap_protocol_clear(unit, protocol_key);`
      Clear the trap action profile for desired protocol and protocol-handling profile

      After using clear, the get function return an empty `trap_gport`.
   b. `bcm_rx_trap_type_destroy(unit, trap_id);`
      Destroy trap, deallocate hardware resources

12.5.2.3 Shell Commands

The following shell commands are available:

- `dbal tbl dump tbl=PROTOCOL_<protocol_type>_TRAP_ACTION_PROFILE`
  Show the table containing trap configurations for a specified protocol type.

  The `protocol_type` value can be `L2CP`, `IGMP`, `ICMP`, `8021X`, `DHCP`, or `ARP_NDP`.
- `dbal tbl dump tbl=PROTOCOL_TRAP_HANDLING_PROFILE`
  Show the table containing all protocol profiles configurations.

12.5.2.4 Application Reference

Examples of protocol traps usage

- **Type:** CINT reference
- **Path:** cint_rx_trap_protocol_traps.c

12.5.3 Programmable Traps

Eight programmable traps are available. Using the programmable traps allows the creation of a customized trap that matches/mismatches/ignores each one of the following qualifiers on the forwarding header:

- DA
- SA
- Ethernet type
- Sub type
- IP protocol
- TCP sequence is zero
- TCP flags
- L4 ports

NOTE: With IP forwarding, Ethernet qualifiers (DA, SA, Ethernet type, subtype) are disregarded.

Set the desired Trap-Action-Profile to occur on trapped packets.

If a packet answers several programmable trap conditions, the action of the trap with the highest forward/snoop strength is selected. If several programmable traps have the same strength, the lowest index trap is selected.

12.5.3.1 SOC Properties

N/A

12.5.3.2 Qualifier Configuration

bcm_rx_trap_prog_config_t

Qualifier	Field Name	Field Type	Description
SA	src_mac_enable	bcm_rx_trap_prog_enable_t	Disable/match/mismatch source MAC qualifier
	src_mac	bcm_mac_t	Source MAC
	src_mac_nof_bits	uint8	Number of MSBs of source MAC to match or mismatch
DA	dest_mac_enable	bcm_rx_trap_prog_enable_t	Disable/match/mismatch destination MAC qualifier
	dest_mac	bcm_mac_t	Destination MAC
	dest_mac_nof_bits	uint8	Number of MSBs of destination MAC to match/mismatch
Ethernet type	ether_type_enable	bcm_rx_trap_prog_enable_t	Disable/match/mismatch Ethernet type qualifier
	ether_type	uint16	Ethernet type
Subtype	sub_type_enable	bcm_rx_trap_prog_enable_t	Disable/match/mismatch sub type qualifier
	sub_type	uint8	Sub type, first byte after link layer header
	sub_type_mask	uint8	Sub type mask
IP protocol	ip_protocol_enable	bcm_rx_trap_prog_enable_t	Disable/match/mismatch IP protocol qualifier
	ip_protocol	uint8	IP protocol

Qualifier	Field Name	Field Type	Description
L4 ports	l4_ports_enable	bcm_rx_trap_prog_enable_t	Disable/match/mismatch L4 ports (src & dest) qualifier
	src_port	bcm_port_t	Source port
	src_port_mask	uint16	Source port mask
	dest_port	bcm_port_t	Destination port
	dest_port_mask	uint16	Destination port mask
TCP flags	tcp_flags_enable	bcm_rx_trap_prog_enable_t	Disable/match/mismatch Ethernet type qualifier
	tcp_flags	uint16	TCP flags
	tcp_flags_mask	uint16	TCP flags mask
TCP sequence	tcp_seq_is_zero_enable	bcm_rx_trap_prog_enable_t	Disable/match/mismatch TCP sequence number is zero qualifier
	trap_gport	bcm_gport_t	Trap-Action-Profile (trap_id, fwd_strength, snp_strength) encoded as gport

`bcm_rx_trap_prog_enable_t (enum)`

- `bcmRxTrapProgDisable = 0`
- `bcmRxTrapProgEnableMatch = 1`
- `bcmRxTrapProgEnableMismatch = 2`

12.5.3.3 Configuration Flow

The configuration flow for programmable traps is as follows:

- Set programmable trap configuration flow:
  - `bcm_rx_trap_type_create(unit, flags, bcmRxTrapUserDefine, &trap_id);`
    Create user-defined trap, allocate hardware resource
  - `bcm_rx_trap_set(unit, trap_id, &trap_config);`
    Set action of trap in hardware
  - `BCM_GPORT_TRAP_SET(trap_gport, trap_id, fwd_strength, snp_strength);`
    Encode trap_id, forward strength and snoop strength as gport
  - `bcm_rx_trap_prog_config_t_init(&prog_config);`
    Initialize configuration struct *before* setting qualifiers and action
    `prog_config.trap_gport = trap_gport`
  - `bcm_rx_trap_prog_set(unit, flags, prog_index, &prog_config);`
    Set the programmable trap qualifiers and action
- Get programmable trap configuration flow:
  - `bcm_rx_trap_prog_get(unit, prog_index, &prog_config);`
    Get programmable trap qualifiers and action
- Clear programmable trap configuration flow:
  - `bcm_rx_trap_prog_config_t_init(&prog_config);`
    Initialize configuration struct, all qualifiers are disabled
  - `bcm_rx_trap_prog_set(unit, flags, prog_index, &prog_config);`
    Set the programmable trap qualifiers and action to init values
  - `bcm_rx_trap_type_destroy(unit, trap_id);`
    Destroy trap, deallocate hardware resources

12.5.3.4 Shell Commands

The following shell commands are available.

- `dbal tbl dump tbl=TRAP_INGRESS_PROG`
  Show the table containing all programmable trap configurations.
- `dbal ENTry Get TaBLe=TRAP_INGRESS_PROG PROG_PROFILE=<index>`
  Show programmable trap configuration for a specific index.

12.5.3.5 Application Reference

Examples of programmable traps usage:

- **Type:** CINT reference
- **Path:** `cint_rx_trap_programmable_traps.c`

12.5.4 ETPP MTU Traps

ETPP MTU traps allow trapping packets with an MTU violation.

For the LIF-based MTU stages, if the MTU check is enabled, the MTU profile is set as follows:

- If the LIF or RIF type has an MTU profile field (LIF or RIF property), the MTU profile is set based on the LIF or RIF profiles found in the database.
- If no MTU profile field is present, the MTU profile is set according to the port configuration.
- The Compressed Layer Type is set based on the configuration for the specific LIF or RIF

In the port-based MTU stage , if the MTU check is enabled:

- MTU profile is taken from the port configuration.
- The compressed layer type is set to 7.

NOTE: On initialization, all LIF, RIF, and PORT instances point to profile 0, which can be used as a reserved profile to do nothing, or can be the largest MTU value that is used in the system (that is, the default profile).

NOTE: The MTU check includes a CRC of 4 bytes in the packet length.

To configure the MTU per LIF/RIF:

1. Apply the desired action, such as drop or recycle, on trapped packets.
2. Manage trapping for packets with a specific layer type.
3. Set the Trap-Action-Profile and MTU threshold for violation.
   Eight configurable MTU profiles for LIF/RIF are available.

NOTE: The number of profiles available per LIF/RIF might decrease per LIF/RIF type. (Upon using LIF/RIF creation APIs and setting properties, the number of bits allocated for an MTU profile can change from three to two.)

To configure the MTU per port:

1. Apply the desired action, such as drop or recycle, on trapped packets.
2. Set Trap-Action-Profile and MTU threshold for violation.
   Eight MTU profiles are available.

3. If both PORT and LIF/RIF MTU traps are configured with the same strengths and a packet hits both, the LIF/RIF MTU trap is executed.

NOTE: For Ethernet encapsulation, the MTU check is handled in a special manner to allow a different MTU configuration (MTU value and trap) for different LIF stacks. Because the MTU configuration is accessed using a combination of MTU_PROFILE (3b) and compressed layer type (3b), the MTU_PROFILE is taken from the ETH LIF and the compressed layer type is taken from the LIF above it in the stack (for example, IP or MPLS).

Example:

LIF Stack	MPLS: cmp_layer_type = 2	IP: cmp_layer_type = 3
	ARP: mtu_profile = 1	
MTU configuration profile	10 {1, 2}	11 {1, 3}

12.5.4.1 SOC Properties

N/A

12.5.4.2 Configuration Flow

Structs

Struct	Field (and Field Type)	Value
`bcm_switch_control_key_t`	type (`bcm_switch_control_t`)	bcmSwitchLinkLayerMtuFilter
	index (int)	Indicates the layer type. `bcm_field_layer_type_t` ENUM must be used. Valid values: ■ `bcmFieldLayerTypeEth` ■ `bcmFieldLayerTypeIp4` ■ `bcmFieldLayerTypeIp6` ■ `bcmFieldLayerTypeArp` ■ `bcmFieldLayerTypeMpls` ■ `bcmFieldLayerTypePppoe` ■ `bcmFieldLayerTypeSrv6Endpoint` ■ `bcmFieldLayerTypeSrv6Beyond`
`bcm_switch_control_info_t`	value (int)	Indicates the compressed layer type
`flags`	uint32	`bcm_rx_mtu_profile_set` and `bcm_rx_mtu_profile_get` supported flags: ■ `BCM_RX_MTU_RIF` ■ `BCM_RX_MTU_LIF` ■ `BCM_RX_MTU_PORT`
`bcm_rx_mtu_profile_key_t`	cmp_layer_type(uint8)	Compressed layer type Used only for LIF/RIF configuration. For port configuration, the compressed layer type is default 7, and does not need to be supplied.
	mtu_profile (uint8)	User-determined number
`bcm_rx_mtu_profile_value_t`	mtu_val (uint32)	MTU threshold value
	trap_gport (bcm_gport_t)	Encoding Trap-Action-Profile (`trap_id`, `fwd_strength`, `snp_strength`) as gport

Struct	Field (and Field Type)	Value
bcm_rx_mtu_config_t	flags (uint32)	BCM_RX_MTU_RIF = MTU configuration per RIF BCM_RX_MTU_LIF = MTU configuration per LIF BCM_RX_MTU_PORT = MTU configuration per Out-Port
	intf (bcm_if_t)	RIF interface
	gport (bcm_gport_t)	Contains either port or gport for LIF
	mtu_profile (uint8)	User-determined number

The configuration flow is as follows:

- Set MTU ETPP trap configuration flow:
  - `bcm_rx_trap_type_create(unit, flags, bcmRxTrapEgTxUserDefine, &trap_id);`
    Allocate hardware resource for action
  - `bcm_rx_trap_config_t_init(&trap_config);`
    Initialize trap configuration struct before setting desired action.
  - `bcm_rx_trap_set(unit, trap_id, &trap_config);`
    Set action of trap in hardware
  - `BCM_GPORT_TRAP_SET(trap_gport, trap_id, fwd_strength, snp_strength);`
    Encode `trap_id`, forward strength and snoop strength as gport.
  - `bcm_switch_control_indexed_set(unit, key, info);`
    Map layer type to compressed layer type for MTU trapping. Indicates the layer type of the LIF/RIF that is supplied to the MTU configuration structure. Relevant only for MTU per LIF/RIF.

    `key.type = bcmSwitchLinkLayerMtuFilter`

    `key.index = bcm_field_layer_type_t` (for example, bcmFieldLayerTypeIp4, bcmFieldLayerTypeArp)

    info.value = 0-6 (User chosen number for the mapping of layer type)
  - `bcm_rx_mtu_profile_key_t mtu_key;`
  - `bcm_rx_mtu_profile_key_t_init(& mtu_key);`

    `mtu_key.mtu_profile`

    `mtu_key.cmp_layer_type = info.value` (Only for LIF/RIF, irrelevant for PORT)
  - `bcm_rx_mtu_profile_value_t mtu_value;`
  - `bcm_rx_mtu_profile_value_t_init(&mtu_value);`

    `mtu_value.trap_gport = trap_gport;`

    `mtu_value.mtu_val` = User supplied value (MTU threshold of trap, max:12287)

    `flags = mtu_config.flags;`
  - `bcm_rx_mtu_profile_set(unit, flags, &mtu_key, &mtu_value);`
    Set Trap-Action-Profile and MTU threshold to the key combination.
  - `bcm_rx_mtu_config_t mtu_config;`
    Indicate RIF, LIF, or PORT for MTU trap configuration

    `mtu_config.gport` = LIF/PORT (`bcm_gport_t` or physical port, decided by flag);

    `mtu_config.intf` = RIF;

    `mtu_config.mtu_profile = mtu_key.mtu_profile;`
  - `bcm_rx_mtu_set(unit, &mtu_config);`
    Set MTU profile to the LIF, RIF, or PORT

- Get MTU ETPP trap configuration flow:
  - `bcm_switch_control_indexed_get(unit, key, &info);`
    Get mapping of forward layer type to compressed forward layer type for MTU trapping. Relevant only for MTU per LIF or RIF.
  - `bcm_rx_mtu_config_t mtu_config;`
  - Indicate RIF, LIF, or PORT for MTU trap configuration

    `mtu_config.gport` = LIF/PORT (`bcm_gport_t` or physical port, decided by flag);

    `mtu_config.intf` = RIF
  - `bcm_rx_mtu_get(unit, &mtu_config);`
    Get MTU profile of LIF/RIF/PORT
  - `bcm_rx_mtu_profile_key_t mtu_key;`
  - `bcm_rx_mtu_profile_key_t_init(&mtu_key);`

    `mtu_key.mtu_profile = mtu_config.mtu_profile;`

    `mtu_key.cmp_layer_type = info.value` (only for LIF/RIF, irrelevant for PORT);
  - `bcm_rx_mtu_profile_value_t mtu_value;`
  - `bcm_rx_mtu_profile_value_t_init(&mtu_value)`
  - `flags = mtu_config.flags;`
  - `bcm_rx_mtu_profile_get(unit, flags, &mtu_key, &mtu_value);`
    Get the Trap-Action-Profile and MTU threshold.
  - `trap_id = BCM_GPORT_TRAP_GET_ID(mtu_config.trap_gport);`
    Decode `trap_id` from trap gport
  - `bcm_rx_trap_get(unit, trap_id, &trap_config);`
    Get action of trap configured from hardware
- Clear MTU ETPP trap configuration flow:
  - `bcm_switch_control_indexed_get(unit, key, &info)`
    Get mapping of forward layer type to compressed layer type for MTU trapping. Relevant only for MTU per LIF/RIF.
  - `bcm_rx_mtu_config_t mtu_config;`
    Indicate RIF/LIF/PORT for MTU trap configuration

    `mtu_config.gport` = LIF/PORT(`bcm_gport_t`/physical port, decided by flag);;

    `mtu_config.intf` = RIF;
  - `bcm_rx_mtu_get(unit, &mtu_config)`
    Get MTU profile for the LIF/RIF/PORT
  - `bcm_rx_mtu_profile_key_t mtu_key;`

    `mtu_key.mtu_profile = mtu_config.mtu_profile;`

    `mtu_key.cmp_layer_type = info.value` (Only for LIF/RIF, irrelevant for PORT);
  - `flags = mtu_config.flags;`
  - `bcm_rx_mtu_profile_get(unit, flags, &mtu_key, &mtu_value);`
    Get Trap-Action-Profile and MTU threshold of trap
  - `trap_id = BCM_GPORT_TRAP_GET_ID(mtu_value.trap_gport)`
    Decode trap_id from trap gport
  - `bcm_rx_mtu_config_t mtu_config_clear;`
  - `bcm_rx_mtu_config_t_init(&mtu_config_clear);`
    Initialize configuration struct to default values

    Indicate RIF/LIF/PORT to clear its MTU Profile

- `bcm_rx_mtu_set(unit, &mtu_config_clear)`
- `bcm_rx_mtu_profile_key_t mtu_key_clear;`
- `bcm_rx_mtu_profile_key_t_init(&mtu_key_clear);`
  `mtu_key_clear.mtu_profile = mtu_config.mtu_profile;`
  `mtu_key_clear.cmp_layer_type = info.value` (Only for LIF/RIF, irrelevant for PORT);
- `bcm_rx_mtu_profile_value_t mtu_value;`
- `bcm_rx_mtu_profile_value_t_init(&mtu_value);`
- `flags = mtu_config_clear.flags;`
- `bcm_rx_mtu_profile_set(unit, flags, &mtu_key_clear, &mtu_value);`
  Clear Trap-Action-Profile and MTU threshold of trap
- `bcm_rx_trap_type_destroy(unit, trap_id)`
  Destroy trap, deallocate HW resources

12.5.4.3 Shell Commands

The following shell command is available:

- `dbal tbl dump tbl=ETPP_TRAP_MTU_MAP`
  Shows the table containing all MTU trap configurations.

12.5.4.4 Application Reference

Examples of LIF MTU trap usage

- **Type**: CINT reference
- **Path**: `cint_rx_trap_etpp_mtu_trap_lif.c`

Examples of RIF MTU trap usage

- **Type**: CINT reference
- **Path**: `cint_rx_trap_etpp_mtu_trap_rif.c`

Examples of port MTU trap usage

- **Type**: CINT reference
- **Path**: `cint_rx_trap_etpp_mtu_trap_port.c`

12.5.5 MTU Extended Traps

MTU traps configured on certain LIFs can be used in an extended mode to allow MTU value configuration per LIF, thus allowing more MTU values to be compared than in the default mode.

Supported LIFs include the following:

- PPPoE

  The MTU Extended PPPoE is enabled by the `bcmSwitchControlMtuExtendedPppoe` switch control.

  In MTU Extended PPPoE, only one Trap-Action-Profile is configured to be applied on all PPPoE LIFs with an MTU violation.

  This Trap-Action-Profile is configured using the `bcmRxTrapEgTxMtuExtendedPppoe` application trap.
- L3 egress ARP

  The MTU Extended L3 egress ARP is enabled by the `bcmSwitchControlMtuExtendedL3EgressArp` switch control.

  In MTU Extended L3 Egress ARP, only one Trap-Action-Profile is configured to be applied on all ARP_SA_tunneling LIFs with an MTU violation.

  This Trap-Action-Profile is configured using the `bcmRxTrapEgTxMtuExtendedL3EgressArp` application trap.

12.5.5.1 Configuration Flow

Setting the MTU Extended PPPoE trap configuration involves the following APIs:

- `bcm_switch_control_set(unit, bcmSwitchControlMtuExtendedPppoe, 1);`
  Enable MTU Extended PPPoE.
- `bcm_rx_trap_type_create(unit, flags, bcmRxTrapEgTxUserDefine, &trap_id);`
  Allocate a hardware resource for action.
- `bcm_rx_trap_config_t_init(&trap_config);`
  Initialize a trap configuration struct before setting the desired action.
- `bcm_rx_trap_set(unit, trap_id, &trap_config);`
  Set the action of trap in hardware.
- `BCM_GPORT_TRAP_SET(trap_gport, trap_id, fwd_strength, snp_strength);`
  Encode trap_id, forward strength, and snoop strength as gport.
- `bcm_rx_trap_action_profile_set(unit, 0, bcmRxTrapEgTxMtuExtendedPppoe, trap_gport);`
  Set the Trap-Action-Profile for MTU Extended PPPoE trap.
- `bcm_rx_mtu_config_t mtu_config;`
  Indicate the RIF or LIF for MTU trap configuration
  - `mtu_config.gport` = LIF;
  - `mtu_config.intf` = RIF;
  - `mtu_config.mtu` = mtu_value;
- `bcm_rx_mtu_set(unit, &mtu_config);`
  Set the MTU value to the LIF/RIF.

12.5.5.2 Application Reference

Example of MTU Extended PPPoE configuration

- Type: CINT reference
- Path: `cint_rx_trap_etpp_mtu_extended_pppoe.c`

Example of MTU Extended L3 Egress ARP configuration

- Type: CINT reference
- Path: `cint_rx_trap_etpp_mtu_extended_l3_egress_arp.c`

12.5.6 LIF Traps

Using the APIs of LIF traps, the user can configure the forward action per LIF.

Eight profiles are available for ingress configuration, each profile is a Trap-Action-Profile (Trap-Code, forward strength, and snoop strength). One profile is reserved as default, and seven other profiles are configurable.

Four profiles are available for the ETPP configuration, one profile is reserved as default, and three other profiles are configurable. Two profiles can configure a Trap-Action-Profile and another one can configure a Mirror command.

12.5.6.1 SOC Properties

N/A

12.5.6.2 Configuration Flow

Table 69: bcm_rx_trap_lif_config_t

Field (and Field Type)	Value
`lif_type (bcm_rx_trap_lif_type_t )`	bcmRxTrapLifTypeInlif bcmRxTrapLifTypeInRif bcmRxTrapLifTypeOutLif bcmRxTrapLifTypeOutRif
`rif_intf (bcm_if_t)`	RIF interface
`lif_gport (bcm_gport_t)`	LIF gport
`action_gport (bcm_gport_t)`	Encoding Trap-Action-Profile Trap-Action-Profile or mirror_cmd(only for ETPP) to configure.

Flow

- Set LIF trap, configuration of Trap-Action-Profile flow:
  - `bcm_rx_trap_type_create(unit, flags, trap_type, &trap_id);`
    For Ingress, `trap_type=bcmRxTrapUserDefine`
    For ETPP, `trap_type=bcmRxTrapEgTxUserDefine`
    Allocate hardware resource for action
  - `bcm_rx_trap_config_t_init(&trap_config);`
    Initialize trap configuration struct before setting desired action
  - `bcm_rx_trap_set(unit, trap_id, &trap_config);`
    Set action of trap in hardware
  - `BCM_GPORT_TRAP_SET(trap_gport, trap_id, fwd_strength, snp_strength);`
    Encode `trap_id`, forward strength and snoop strength as gport

  - `bcm_rx_trap_lif_config_t lif_config;`
    Indicate RIF/LIF for profile configuration
    `lif_config.action_gport = trap_gport;`
  - `bcm_rx_trap_lif_set(unit, flags, &lif_config);`
    Set Trap-Action-Profile for LIF/RIF
- Set LIF trap, configuration of mirror command flow:
  - `bcm_mirror_destination_create(unit, &mirror_dest)`
    Allocate hardware resource and configure the mirror command
  - `bcm_rx_trap_lif_config_t lif_config;`
    Indicate RIF/LIF for profile configuration
    lif_config.action_gport = mirror_dest.mirror_dest_id;
  - `bcm_rx_trap_lif_set(unit, flags, &lif_config);`
    Set mirror command for LIF/RIF
- Get LIF trap, configuration of Trap-Action-Profile flow:
  - `bcm_rx_trap_lif_config_t lif_config;`
    Indicate RIF/LIF to get profile configuration
  - `bcm_rx_trap_lif_get(unit, &lif_config);`
    Get Trap-Action-Profile for LIF/RIF
  - `trap_id = BCM_GPORT_TRAP_GET_ID(lif_config.action_gport);`
    Decode trap_id from trap gport
  - `bcm_rx_trap_get(unit, trap_id, &trap_config);`
    Get action of trap configured from hardware
- Get LIF trap, configuration of mirror command flow:
  - `bcm_rx_trap_lif_config_t lif_config;`
    Indicate RIF/LIF to get profile configuration
  - `bcm_rx_trap_lif_get(unit, &lif_config);`
    Get mirror command for LIF/RIF
  - `bcm_mirror_destination_get(unit, lif_config.action_gport, &mirror_dest);`
    Get the configuration of the mirror command
- Destroy LIF trap, configuration of Trap-Action-Profile flow:
  - `bcm_rx_trap_lif_get(unit, &lif_config);`
    Get Trap-Action-Profile for LIF/RIF
  - `trap_id = BCM_GPORT_TRAP_GET_ID(lif_config.action_gport);`
    Decode trap_id from trap gport
  - `bcm_rx_trap_type_destroy(unit, trap_id)`
    Destroy trap, deallocate hardware resources
  - `bcm_rx_trap_lif_config_t lif_config_clear;`
  - `bcm_rx_trap_lif_config_t_init(&lif_config_clear);`
    Initialize configuration struct to default values
    Indicate RIF/LIF to clear lif trap profile
    `bcm_rx_trap_lif_set(unit, flags, &lif_config_clear);`
    Clear profile for LIF/RIF
- Destroy LIF trap, configuration of mirror command flow:
  - `bcm_rx_trap_lif_get(unit, &lif_config);`
    Get mirror command for LIF/RIF
  - `bcm_mirror_destination_destroy(unit, lif_config.action_gport);`
    Destroy mirror command
  - `bcm_rx_trap_lif_config_t lif_config_clear;`

- `bcm_rx_trap_lif_config_t_init(&lif_config_clear);`
  Initialize configuration struct to default values
  Indicate RIF/LIF to clear lif trap profile
  `bcm_rx_trap_lif_set(unit, flags, &lif_config_clear);`
  Clear profile for LIF/RIF

12.5.6.3 Shell Commands

The following shell command is available:

- `dbal tbl dump tbl=TRAP_INGRESS_LIF_ACTION_PROFILE/ETPP_TRAP_LIF_ACTION_PROFILE`
  Show the table containing all ingress and ETPP LIF trap configurations

12.5.6.4 Application Reference

Examples of LIF trap usage

- **Type:** CINT reference
- **Path:** `cint_rx_trap_lif_traps.c`

12.6 API Descriptions

12.6.1 bcm_rx_trap_*

API name: `bcm_rx_trap_type_create(unit, flags, trap_type, &trap_id);`

Highlights: Create a trap, allocate a trap_id for the required trap_type.

Parameters:

- Input:
  - `unit` (int) – Device number.
  - `flags` (int)
    - For predefined flags=0.
    - For user defined can also be BCM_RX_TRAP_WITH_ID.
      When WITH_ID flag is set, trap_id will hold the desired id as input.
  - `trap_type` (`bcm_rx_trap_t`) – Type of trap to create.
- In/Output:
  - `trap_id` (int) – Returns the hardware id of action profile, i.e entry in action table.

API name: `bcm_rx_trap_set(unit, trap_id, &config);`

Highlights: Configure the Action-Profile for the required `trap_id`.

Parameters:

- Input:
  - `unit` (int) – Device number.
  - `trap_id` (int) – The hardware id of action profile, i.e entry in action table.
  - `config` (`bcm_rx_trap_config_t`) – Action profile configuration.

API name: `bcm_rx_trap_get(unit, trap_id, &config);`

Highlights: Get configuration the Action-Profile for the required `trap_id`.

Parameters:

- Input:
  - `unit` (int) – Device number.
  - `trap_id` (int) – The hardware id of action profile, i.e entry in action table.
- Output:
  - `config` (`bcm_rx_trap_config_t`) – Action profile configuration.

API name: `bcm_rx_trap_type_get(unit, flags, trap_type, &trap_id);`

Highlights: Get the trap_id value (HW code) of a given trap_type.

Parameters:

- Input:
  - `unit` (int) – Device number.
  - `flags` (int) – Reserved for future use.
  - `trap_type` (bcm_rx_trap_t) – Type of trap.
- Output:
  - `trap_id` (int) – The HW id of action profile, i.e entry in action table.

API name: `bcm_rx_trap_type_from_id_get(unit, flags, trap_id, &trap_type);`

Highlights: Get the trap_type of a given trap_id (HW code).

Parameters:

- Input:
  - `unit` (int) – Device number.
  - `flags` (int) – Reserved for future use.
  - `trap_id` (int) – The HW id of action profile, i.e entry in action table.
- Output:
  - `trap_type` (bcm_rx_trap_t) – Type of trap.

API name: `bcm_rx_trap_type_destroy(unit, trap_id);`

Highlights: Destroy trap, deallocate `trap_id` and clean the Action-Profile.

Parameters:

- Input:
  - `unit` (int) – Device number.
  - `trap_id` (int) – The hardware id of action profile, that is, entry in action table.

API name: `bcm_rx_trap_config_t_init(&config);`

Highlights: Initialize the Action-Profile configuration.

Parameters:

- Output:
  - `config` (`bcm_rx_trap_config_t`) – Action profile configuration.

12.6.2 bcm_rx_trap_action_profile_*

API name: `bcm_rx_trap_action_profile_set(unit, flags, trap_type, trap_gport);`

Highlights: Configure the trap action profile for specific trap type.

Parameters:

- Input:
  - `unit` (int) – Device number.
  - `flags` (uint32) – Reserved for future use.
  - `trap_type` (`bcm_rx_trap_t`) – Type of trap to create.
  - `trap_gport` (`bcm_gport_t`) – Trap-Action-Profile: action profile, forward strength and snoop strength encoded as gport.

API name: `bcm_rx_trap_action_profile_get(unit, trap_type, &trap_gport);`

Highlights: Get configuration of trap action profile for specific trap type.

Parameters:

- Input:
  - `unit` (int) – Device number.
  - `trap_type` (`bcm_rx_trap_t`) – Type of trap to get its Trap-Action-Profile.
- Output:
  - `trap_gport` (`bcm_gport_t`) – Trap-Action-Profile: action profile, forward strength and snoop strength encoded as gport.

API name: `bcm_rx_trap_action_profile_clear(unit, trap_type);`

Highlights: Clear configuration of trap action profile for specific trap type.

Parameters:

- Input:
  - `unit` (int) – Device number.
  - `trap_type` (`bcm_rx_trap_t`) – Type of trap to clear its Trap-Action-Profile.

12.6.3 bcm_rx_trap_protocol_*

API name: `bcm_rx_trap_protocol_set(unit, &protocol_key, trap_gport);`

Highlights: Configure the trap action profile for required protocol key.

Parameters:

- Input:
  - `unit (int)` – Device number
  - `protocol_key` (`bcm_rx_trap_protocol_key_t`)
    - `protocol_type` (`bcm_rx_trap_t`) – Protocol type
    - `protocol_trap_profile` – Protocol handling profile (2b)
    - `trap_args` – Argument suitable for protocol type
      - L2CP – MAC's 6 LSBs (0 to 63)
      - ICMP – ICMP types (0 to 31)
  - `trap_gport` (`bcm_gport_t`) – Trap-Action-Profile: action profile, forward strength and snoop strength encoded as gport.

API name: `bcm_rx_trap_protocol_get(unit, &protocol_key, &trap_gport);`

Highlights: Get configuration the trap action profile for required protocol key.

Parameters:

- Input:
  - `unit (int)` – Device number
  - `protocol_key` (`bcm_rx_trap_protocol_key_t`)
    - `protocol_type` (`bcm_rx_trap_t`) – Protocol type
    - `protocol_trap_profile` – Protocol handling profile (2b)
    - `trap_args` – Argument suitable for protocol type
      - L2CP – MAC's 6 LSBs (0 to 63)
      - ICMP – ICMP types (0 to 31)
- Output:
  - `trap_gport` (`bcm_gport_t`) – Trap-Action-Profile: action profile, forward strength and snoop strength encoded as gport.

API name: `bcm_rx_trap_protocol_clear(unit, &protocol_key);`

Highlights: Clear configuration the trap action profile for required protocol key.

Parameters:

- Input:
  - `unit (int)` – Device number
  - `protocol_key` (`bcm_rx_trap_protocol_key_t`) -
    - `protocol_type` (`bcm_rx_trap_t`) – Protocol type
    - `protocol_trap_profile` – Protocol handling profile (2b)
    - `trap_args` – Argument suitable for protocol type
      - L2CP – MAC's 6 LSBs (0 to 63)
      - ICMP – ICMP types (0 to 31)

API name: `bcm_rx_trap_protocol_key_t_init(&key_config);`

Highlights: Initialize the protocol trap key configuration struct.

Parameters:

- Output:
  - `key_config (bcm_rx_trap_protocol_key_t)` – Protocol trap key configuration struct initialized.

API name: `bcm_rx_trap_protocol_profiles_set(unit, port, &protocol_profiles);`

Highlights: Configure protocol trap profiles for specific LIF/port.

Parameters:

- Input:
  - `unit (int)` – Device number.
  - `port` (`bcm_gport_t`) – Either LIF (encoded as gport) or port
  - `protocol_profiles` (`bcm_rx_trap_protocol_profiles_t`):
    - `l2cp_trap_profile (uint8)`
    - `icmpv4_trap_profile` (uint8, used for ICMP types for IPv4)
    - `icmpv6_trap_profile` (uint8, used for ICMP types for IPv6)
    - `arp_trap_profile (uint8)`
    - `igmp_trap_profile (uint8)`
    - `dhcp_trap_profile (uint8)`
    - `non_auth_8021x_trap_profile (uint8)`

API name: `bcm_rx_trap_protocol_profiles_get(unit, port, &protocol_profiles);`

Highlights: Configure protocol trap profiles for specific LIF/port.

Parameters:

- Input:
  - `unit (int)` – Device number.
  - `port` (`bcm_gport_t`) – Either LIF (encoded as gport) or port.
- Output:
  - `protocol_profiles` (`bcm_rx_trap_protocol_profiles_t`)
    - `l2cp_trap_profile (uint8)`
    - `icmpv4_trap_profile` (uint8, used for ICMP types for IPv4)
    - `icmpv6_trap_profile` (uint8, used for ICMP types for IPv6)
    - `arp_trap_profile (uint8)`
    - `igmp_trap_profile (uint8)`
    - `dhcp_trap_profile (uint8)`
    - `non_auth_8021x_trap_profile (uint8)`

API name: `bcm_rx_trap_protocol_profiles_t_init(&profiles_config);`

Highlights: Initialize the protocol trap profiles configuration struct.

Parameters:

- Output: `profiles_config (bcm_rx_trap_protocol_profiles_t)` – Protocol trap profiles configuration struct initialized.

API name: `bcm_rx_trap_protocol_interface_profiles_set(unit, intf, &protocol_profiles);`

Highlights: Configure protocol trap profiles for a specific RIF interface.

Parameter (Input):

- `unit (int)` – Device number.
- `intf (bcm_ift_t)` – RIF Interface
- `protocol_profiles (bcm_rx_trap_protocol_profiles_t)`:
  - `l2cp_trap_profile` (uint8)
  - `icmpv4_trap_profile` (uint8, used for ICMP types for IPv4)
  - `icmpv6_trap_profile` (uint8, used for ICMP types for IPv6)
  - `arp_trap_profile` (uint8)
  - `igmp_trap_profile` (uint8)
  - `dhcp_trap_profile` (uint8)
  - `non_auth_8021x_trap_profile` (uint8)

API name: `bcm_rx_trap_protocol_interface_profiles_get(unit, intf, &protocol_profiles);`

Highlights: Get the protocol trap profiles for a specific RIF interface.

Parameters:

- Input:
  - `unit (int)` – Device number.
  - `intf (bcm_if_t)` – RIF interface.
- Output:
  - `protocol_profiles (bcm_rx_trap_protocol_profiles_t)`:
    - `l2cp_trap_profile` (uint8)
    - `icmpv4_trap_profile` (uint8, used for ICMP types for IPv4)
    - `icmpv6_trap_profile` (uint8, used for ICMP types for IPv6)
    - `arp_trap_profile` (uint8)
    - `igmp_trap_profile` (uint8)
    - `dhcp_trap_profile` (uint8)
    - `non_auth_8021x_trap_profile` (uint8)

12.6.4 bcm_rx_trap_prog_*

API name: `bcm_rx_trap_prog_set(unit, flags, prog_index, &prog_config);`

Highlights: Configure programmable trap.

Parameters:

- Input:
  - `unit (int)` – Device number.
  - `flags (uint32)` – Reserved for future use.
  - `prog_index (uint8)` – Index of programmable trap to configure.
  - `prog_config` (`bcm_rx_trap_prog_config_t`) – Programmable trap configuration, hold the qualifiers to configure, each qualifier state (match/mismatch/ignore) and Trap-Action-Profile encoded as gport.

API name: `bcm_rx_trap_prog_get(unit, prog_index, &prog_config);`

Highlights: Get configuration of programmable trap.

Parameters:

- Input:
  - `unit (int)` – Device number.
  - `prog_index (uint8)` – Index of programmable trap to configure.
- Ouput:
  - `prog_config` (`bcm_rx_trap_prog_config_t`) – Programmable trap configuration, hold the qualifiers to configure, each qualifier state (match/mismatch/ignore) and Trap-Action-Profile encoded as gport.

API name: `bcm_rx_trap_prog_config_t_init(&prog_config);`

Highlights: Initialize programmable trap configuration struct.

Parameters:

- Input:
  - `prog_config` (`bcm_rx_trap_prog_config_t`) – Programmable trap configuration. Initialize all qualifiers as disabled.

12.6.5 bcm_rx_mtu_*

API name: `bcm_rx_mtu_set(unit, &mtu_config)`

Highlights: Set MTU value and Trap-Action-Profile per LIF/RIF/port.

Parameters:

- Input:
  - `unit (int)` – Device number.
  - `mtu_config` (`bcm_rx_mtu_config_t`) – MTU trap configuration.

API name: `bcm_rx_mtu_get(unit, &mtu_config)`

Highlights: Get MTU value and Trap-Action-Profile per LIF/RIF/port.

Parameters:

- Input:
  - `unit (int)` – Device number.
- Output:
  - `mtu_config` (`bcm_rx_trap_mtu_config_t`) – MTU trap configuration.

API name: `bcm_rx_mtu_config_t_init(&mtu_config);`

Highlights: Initialize the MTU trap configuration struct.

Parameters:

- Output:
  - `mtu_config (bcm_rx_mtu_config_t)` – MTU trap configuration struct initialized.

12.6.6 bcm_rx_trap_lif_*

API name: `bcm_rx_trap_lif_set(unit, flags, &lif_config)`

Highlights: Set LIF trap profile configuration per LIF/RIF.

Parameters:
- Input:
  - `unit (int)` – Device number.
  - `flags (uint32)` – Reserved for future use.
  - `lif_config (bcm_rx_trap_lif_config_t)` – LIF trap configuration.

API name: `bcm_rx_trap_lif_get(unit, &lif_config)`

Highlights: Get LIF trap profile configuration per LIF/RIF.

Parameters:
- Input:
  - `unit (int)` – Device number.
- Output:
  - `lif_config (bcm_rx_trap_lif_config_t)` – LIF trap configuration.

API name: `bcm_rx_trap_lif_config_t_init(&lif_config);`

Highlights: Initialize the LIF trap configuration struct.

Parameters:
- Output:
  - `lif_config (bcm_rx_trap_lif_config_t)` – LIF trap configuration struct initialized

12.7 Device-Specific Traps

This section contains trap information specific to the BCM88690, BCM88800, and BCM88480 devices.

12.7.1 Trap Information Specific to the BCM88690 Device (Jericho2)

12.7.1.1 Ingress Predefined Traps

Table 70: List of IRPP Predefined Traps

bcmRxTrap…	Description	HW Trap ID	Created on Init
PortNotVlanMember	Initial VID membership error	0x10	✓
StpStateBlock	Packet ingresses a blocked port	0x27	✓
StpStateLearn	Packet ingresses a port in a LEARN state	0x28	✓
IpCompMcInvalidIp	L2 compatible MC, but IP does not match	0x18	—
MyMacAndIpDisabled	Reached My-MAC over IP, but routing is disabled for InRIF	0x14	✓
MyMacAndMplsDisable	Reached My-MAC over MPLS, but MPLS is disabled for InRIF	0x15	—
ArpReply	Terminated My-MAC over ARP	0x16	—
MyMAcAndUnknownL3	Terminated My-MAC over Unknown L3	0x17	—
MplsTerminationFail	MPLS already terminated twice by label range match, additional labels may not be terminated by Label range match	0x2e	—
MplsUnexpectedBos	Terminated MPLS label indicates that another MPLS header follows, but following label is BOS	0x21	—
MplsUnexpectedNoBos	Terminated MPLS label doesn't indicate there's a following header, but following label is not BOS	0x22	—
L2DiscardMacsaFwd	Triggered by SA procedure (sa_drop), configured by default to FWD the packet	0x8c	—
L2DiscardMacsaDrop	Triggered by SA procedure (sa_drop), configured by default to drop the packet	0x8d	—
L2DiscardMacsaTrap	Triggered by SA procedure (sa_drop), configured by default to redirect the packet to CPU	0x8e	—
L2DiscardMacsaSnoop	Triggered by SA procedure (sa_drop), configured by default to snoop the packet	0x8f	—
L2Learn0	Triggered by SA procedure (sa_not_found), configured by default to FWD the packet	0x90	✓
L2Learn1	Triggered by SA procedure (sa_not_found), configured by default to drop the packet	0x91	✓
L2Learn2	Triggered by SA procedure (sa_not_found), configured by default to redirect the packet to CPU	0x92	✓
L2Learn3	Triggered by SA procedure (sa_not_found), configured by default to snoop the packet	0x93	✓
L2DlfFwd	Triggered by Ethernet default procedure (da_not_found)	0x94	—
L2DlfDrop	Triggered by Ethernet default procedure (da_not_found)	0x95	—
L2DlfTrap	Triggered by Ethernet default procedure (da_not_found)	0x96	—
L2DlfSnoop	Triggered by Ethernet default procedure (da_not_found)	0x97	—
SameInterface	Source interface is equal to destination interface (hair-pin)	0xa3	✓
UcLooseRpfFail	Forwarding Code is IPv4 UC and RPF FEC pointer valid is not set	0xa6	✓
MplsUnknownLabel	In case the MPLS lookup did not hit any result	0xc	✓

Table 70: List of IRPP Predefined Traps (Continued)

bcmRxTrap…	Description	HW Trap ID	Created on Init
Ipv6HopCount0	Terminated IPv6 header hop count (TTL) equals to zero	0x4b	—
Ipv6HopCount1	Terminated IPv6 header hop count (TTL) equals to one	0x4c	—
TcpSnFlagsZero	TCP packet, L4 sequence-number and flags are both zero	0x83	—
TcpSnZeroFlagsSet	TCP packet, L4 sequence-number is zero and either FIN/URG/PSH Flags are set	0x84	—
TcpSynFin	TCP packet, SYN and FIN Flags are both set	0x85	—
TcpEqualPorts	TCP packet, L4 source port equals destination port	0x86	—
TcpFragmentIncompleteHeader	TCP packet, L3 is IPv4 and IP-header fragmented with fragment-offset zero, and IP-header (total-length - 4*IHL) is less than 20	0x87	—
UdpEqualPorts	UDP packet, L4 source port equals destination port	0x89	—
IcmpDataGt576	ICMP packet, L3 is IPv4 and IP-header (total-length - 4*IHL) is greater than 576B or L3 is IPv6 and IP-header payload-length is greater than 576B	0x8a	—
IcmpFragmented	ICMP packet and IP-header is fragmented	0x8b	—
FailoverFacilityInvalid	Both destination-0-valid and destination-1-valid are not set	0xc0	—
UcStrictRpfFail	UC-RPF-Mode is *strict* and OutRIF is not equal to packet InRIF	0xc2	✓
McExplicitRpfFail	MC-RPF-Mode is e*xplicit* and RPF-Entry. Expected-InRIF is not equal to packet InRIF	0xc3	✓
McUseSipRpfFail	MC-RPF-Mode is *Use-SIP-WITH-ECMP* and Out-RIF is not equal to In-RIF	0xc4	✓
McUseSipMultipath	MC-RPF-Mode is *Use-SIP* and RPF-ECMP-Group-Size > 1	0xc5	—
IcmpRedirect	ICMP-Redirect is enabled, Forwarding-Code is IPv4|6-UC, and packet InRIF is equal to FEC-Entry.OutRIF	0xc6	—
OamEthAccelerated	Accelerated Ethernet OAM	0xe0	✓
FcoeSrcIdMismatchSa	FCoE packet, where FC.SrC mismatches Eth.SA	0xae	—
OamY1731MplsTp	Accelerated MPLS-TP OAM	0xe1	✓
OamY1731Pwe	Accelerated PWE OAM	0xe2	✓
OamBfdIpv4	BFD over IPv4 including single hop, multi-hop and micro-BFD	0xe3	✓
OamBfdMpls	BFD over LSP, including IPv4 and IPv6	0xe4	✓
OamBfdPwe	BFD over PWE	0xe5	✓
OamLevel	OAM packet with an MD level below the highest MEP on active side	0xa2	✓
OamPassive	OAM packet equal to or below the highest MEP level on passive side	0xac	✓
1588	IEEE 1588 protocol packet	0x98	—
Failover1Plus1Fail	One plus one protected LIF and failover status is down	0x2c	✓
DfltDroppedPacket	Default action to drop packet	0xaf	✓
DfltRedirectToCpuPacket	Default action to redirect packet to CPU	0xb3	✓
SnoopOamPacket	OAM snoop	0xfe	✓
1588Discard	IEEE 1588 protocol packet, configured by default to drop the packet	0x99	✓
1588Accepted	IEEE 1588 protocol packet, configured by default to forward the packet	0x9a	—
EgTxFieldSnoop0	Trap code reserved for mapping to snoop command 0	0x30	✓
EgTxFieldSnoop1	Trap code reserved for mapping to snoop command 1	0x5b	✓
EgTxFieldSnoop2	Trap code reserved for mapping to snoop command 2	0x5c	✓
EgTxFieldSnoop3	Trap code reserved for mapping to snoop command 3	0x5d	✓
EgTxFieldSnoop4	Trap code reserved for mapping to snoop command 4	0x5e	✓

Table 70: List of IRPP Predefined Traps (Continued)

bcmRxTrap…	Description	HW Trap ID	Created on Init
EgTxFieldSnoop5	Trap code reserved for mapping to snoop command 5	0x5f	✓
EgTxFieldSnoop6	Trap code reserved for mapping to snoop command 6	0x31	✓
EgTxFieldSnoop7	Trap code reserved for mapping to snoop command 7	0x61	✓
EgTxFieldSnoop8	Trap code reserved for mapping to snoop command 8	0x62	✓
EgTxFieldSnoop9	Trap code reserved for mapping to snoop command 9	0x63	✓
EgTxFieldSnoop10	Trap code reserved for mapping to snoop command 10	0x64	✓
EgTxFieldSnoop11	Trap code reserved for mapping to snoop command 11	0x65	✓
EgTxFieldSnoop12	Trap code reserved for mapping to snoop command 12	0x66	✓
EgTxFieldSnoop13	Trap code reserved for mapping to snoop command 13	0x67	✓
EgTxFieldSnoop14	Trap code reserved for mapping to snoop command 14	0xa4	✓
EgTxFieldSnoop15	Trap code reserved for mapping to snoop command 15	0xa5	✓
OamBfdIpv6	BFD over IPv6 including single hop, multi-hop and micro-BFD	0xe8	✓
BfdOamDownMEP	Non-accelerated default ingress trap for OAM and BFD	0xfd	✓
MplsTunnelTerminationTtl0	Terminated MPLS header TTL equals 0	0x23	—
MplsTunnelTerminationTtl1	Terminated MPLS header TTL equals 1	0x24	—
MplsForwardingTtl0	MPLS forwarding trap with TTL equals 0	0x81	—
MplsForwardingTtl1	MPLS forwarding trap with TTL equals 1	0x82	—
OamReflector	OAM Down MEP LBM reply through reflector	0xdf	✓
LinkLayerVlanTagDiscard	Discarded packets according to their VLAN tag configuration	0x6	✓
TerminatedVlanTagDiscard	Discarded terminated packets according to their VLAN tag configuration	0x33	✓
LinkLayerVlanTagAccept	Accepted packets according to their VLAN tag configuration	0x5	—
TerminatedVlanTagAccept	Accepted terminated packets according to their VLAN tag configuration	0x32	—
LinkLayerSaEqualsDa	Source address equals destination address	0x4	—
TerminatedSaEqualsDa	Source address equals destination address on terminated packet	0x3d	—
LinkLayerSaMulticast	Source address is multicast. The trap can be enabled per-port using `bcmPortControlSaMulticastEnable` in `bcm_port_control_set()`	0x3	—
TerminatedSaMulticast	Source address is multicast on terminated packet	0x3c	—
LinkLayerHeaderSizeErr	Parser indicates a header size error, meaning the parsed header size is greater than 144B. **NOTE:** The last parsed layer might exceed 144B, and a trap is not raised if the exceeded field is not used by the parser.	0x0	—
TerminatedHeaderSizeErr	Parser indicates terminated header size error, meaning the parsed header size is greater than 144B. **NOTE:** The last parsed layer might exceed 144B, and a trap is not raised if the exceeded field is not used by the parser.	0x39	—
ForwardingIpv4VersionError	Forwarding IPv4 header version is different than 4	0x68	—
TerminatedIpv4VersionError	Terminated IPv4 header version is different than 4	0x40	—
ForwardingIpv4ChecksumError	Forwarding IPv4 Internet header length is 5 and the checksum over the first 20 bytes doesn't verify	0x69	—
TerminatedIpv4ChecksumError	Terminated IPv4 Internet header length is 5, and the checksum over the first 20 bytes does not verify	0x41	—
ForwardingIpv4HeaderLengthError	Forwarding IPv4 Internet header length is less than 5	0x6a	—
TerminatedIpv4HeaderLengthError	Terminated IPv4 Internet header length is less than 5	0x42	—

Table 70: List of IRPP Predefined Traps (Continued)

bcmRxTrap…	Description	HW Trap ID	Created on Init
ForwardingIpv4TotalLengthError	Forwarding IPv4 header total length is less than 20	0x6b	—
TerminatedIpv4TotalLengthError	Terminated IPv4 header total length is less than 20	0x43	—
ForwardingIpv4HasOptions	Forwarding IPv4 Internet header length is greater than 5	0x6d	—
TerminatedIpv4HasOptions	Terminated IPv4 Internet header length is greater than 5	0x45	—
ForwardingIpv4SipEqualDip	Forwarding IPv4 source IP equals to destination IP	0x6f	—
TerminatedIpv4SipEqualDip	Terminated IPv4 source IP equals to destination IP	0x47	—
ForwardingIpv4DipZero	Forwarding IPv4 destination IP equals to zero	0x70	—
TerminatedIpv4DipZero	Terminated IPv4 destination IP equals to zero	0x48	—
ForwardingIpv4SipIsMc	Forwarding IPv4 source IP is multicast	0x71	—
TerminatedIpv4SipIsMc	Terminated IPv4 source IP is multicast	0x49	—
ForwardingIpv4Ttl0	Forwarding IPv4 header TTL equals to zero	0x6c	—
TerminatedIpv4Ttl0	Terminated IPv4 header TTL equals to zero	0x44	—
ForwardingIpv4Ttl1	Forwarding IPv4 header TTL equals to one	0x6e	—
TerminatedIpv4Ttl1	Terminated IPv4 header TTL equals to one	0x46	—
ForwardingIpv6VersionError	Forwarding IPv6 header Version is different than 6	0x72	—
TerminatedIpv6VersionError	Terminated IPv6 header Version is different than 6	0x4a	—
ForwardingIpv6UnspecifiedDestination	Forwarding IPv6 header DIP is unspecified	0x75	—
TerminatedIpv6UnspecifiedDestination	Terminated IPv6 header DIP is unspecified	0x4d	—
ForwardingIpv6LoopbackAddress	Forwarding IPv6 header SIP or DIP are loopback addresses	0x76	—
TerminatedIpv6LoopbackAddress	Terminated IPv6 header SIP or DIP are loopback addresses	0x4e	—
ForwardingIpv6MulticastSource	Forwarding IPv6 header SIP is multicast	0x77	—
TerminatedIpv6MulticastSource	Terminated IPv6 header SIP is multicast	0x4f	—
ForwardingIpv6NextHeaderNull	Forwarding IPv6 header next protocol is zero	0x78	—
TerminatedIpv6NextHeaderNull	Terminated IPv6 header next protocol is zero	0x50	—
ForwardingIpv6UnspecifiedSource	Forwarding IPv6 header SIP is unspecified	0x79	—
TerminatedIpv6UnspecifiedSource	Terminated IPv6 header SIP is unspecified	0x51	—
ForwardingIpv6LocalLinkDestination	Forwarding IPv6 header DIP bits 127:118 equal to 0x3FA	0x7a	—
TerminatedIpv6LocalLinkDestination	Terminated IPv6 header DIP bits 127:118 equal to 0x3FA	0x52	—
ForwardingIpv6LocalSiteDestination	Forwarding IPv6 header DIP bits 127:118 equal to 0x3FB	0x7b	—
TerminatedIpv6LocalSiteDestination	Terminated IPv6 header DIP bits 127:118 equal to 0x3FB	0x53	—
ForwardingIpv6LocalLinkSource	Forwarding IPv6 header SIP bits 127:118 equal to 0x3FA	0x7c	—
TerminatedIpv6LocalLinkSource	Terminated IPv6 header SIP bits 127:118 equal to 0x3FA	0x54	—
ForwardingIpv6LocalSiteSource	Forwarding IPv6 header SIP bits 127:118 equal to 0x3FB	0x7d	—
TerminatedIpv6LocalSiteSource	Terminated IPv6 header SIP bits 127:118 equal to 0x3FB	0x55	—
ForwardingIpv6Ipv4Compatible Destination	Forwarding IPv6 header DIP bits 127:32 equal to zero	0x7e	—
TerminatedIpv6Ipv4Compatible Destination	Terminated IPv6 header DIP bits 127:32 equal to zero	0x56	—
ForwardingIpv6MulticastDestination	Forwarding IPv6 header DIP is multicast	0x80	—
TerminatedIpv6MulticastDestination	Terminated IPv6 header DIP is multicast	0x58	—
TerminatedCoeFlowControl	Channelized over Ethernet flow-control terminated packet	0x37	—
TerminatedIpv4Fragmented	Terminated IPv4 header is fragmented	0x59	—

Table 70: List of IRPP Predefined Traps (Continued)

bcmRxTrap…	Description	HW Trap ID	Created on Init
TerminatedMplsControlWordTrap	MPLS label with control word and first nibble equals one	0x25	—
TerminatedMplsControlWordDrop	MPLS label with control word and first nibble does not equal one or zero	0x26	—
MplsPreprocessingBosOrTtl	Error of BOS or TTL was identified in MPLS preprocessing	0x1f	—
UnknownDest	Unknown destination	0xa	✓
Default	Default trap, no change to packet's forward action	0x9	✓
EgressTrapped2ndPass	Egress trapped packet on the second pass	0xb	✓
1588User1	IEEE 1588 protocol user trap 1	0x9b	—
1588User2	IEEE 1588 protocol user trap 2	0x9c	—
1588User3	IEEE 1588 protocol user trap 3	0x9d	—
1588User4	IEEE 1588 protocol user trap 4	0x9e	—
1588User5	IEEE 1588 protocol user trap 5	0x9f	—
EgBfdIpv6InvalidUdpChecksum	BFD IPv6 trap for invalid UDP checksum	0xf3	—
OamPerformanceEthAccelerated	Down MEP trapping of LM/DM to OAMP	0xa9	✓
OamPerformanceY1731MplsTp	MPLS-TP trapping of LM/DM to OAMP	0xaa	✓
OamPerformanceY1731Pwe	PWE trapping of LM/DM to OAMP	0xab	✓
ForwardingIpv6Ttl0	Fwd IPv6 TTL is equal to 0	0x73	—
ForwardingIpv6Ttl1	Fwd IPv6 TTL is equal to 1	0x74	—
Srv6Usp	SRv6 USP flow with L4 header over SRv6 header	0x29	—
ItmhError	Trap for cases of ITMH Error	0x12	✓
MplsSpeculativeParseFail	MPLS speculative parsing failure	0xad	—

NOTE: To check the action configured, use the `trap action info id=<trap_id>` diagnostic command.

12.7.1.2 ERPP Application Traps

Table 71: List of ERPP Application Traps

bcmRxTrap…	Description	Created on Init
EgHairPinFilter	Source interface is equal to destination interface (hair-pin)	✓
EgSplitHorizonFilter	Triggered by conditions on incoming and outgoing packet orientation	—
EgUnknownDa[a]	Unknown destination address	—
EgDiscardFrameTypeFilter	Unacceptable frame type on port (tagged, untagged)	—
EgIpmcTtlErr	IPv4 MC packet with invalid TTL	✓
EgIpv4Ttl0	IPv4 header TTL equals to zero	✓
EgIpv4Ttl1	IPv4 header TTL equals to one	✓
EgExcludeSrc	Trap packet of specific source	✓
EgGlemPpTrap	Triggered when the ERPP GLEM lookup does not find an entry and the packet goes through standard processing.	—
EgGlemNonePpTrap	Triggered when the ERPP GLEM lookup does not find an entry and the packet goes through non-standard processing; for example, mirrored or ERPP trapped packets.	—

a. EgUnknownDa is not supported by the BCM88690 A0 device.

12.7.1.3 ETPP Application Traps

Table 72: List of ETPP Application Traps

bcmRxTrap…	Description	Created on Init
EgTxStpStateFail	Triggered by STP state failure	✓
EgTxProtectionPathUnexpected	Unexpected traffic on a protecting path	✓
EgTxPortNotVlanMember	VID membership error	✓
EgTxDiscardFrameTypeFilter	Unacceptable frame type on port (tagged, untagged)	—
EgTxSplitHorizonFilter	Triggered by conditions on incoming and outgoing packet orientation	✓
EgTxLatency	Triggered by latency measurements	—
EgTxMetering	Triggered by meter limitations	—
EgTxGlem	Triggered when ETPP GLEM lookup does not find an entry.	—
EgTxMtuExtendedPppoe	MTU Extended PPPoE trap	—
EgTxMtuExtendedL3EgressArp	MTU Extended L3 Egress ARP trap	—

12.7.2 Trap Information Specific to the BCM88800 Device (Jericho2C)

12.7.2.1 Ingress Predefined Traps

Table 73: List of IRPP Predefined Traps

bcmRxTrap…	Description	HW Trap ID	Created on Init
PortNotVlanMember	Initial VID membership error	0x10	✓
StpStateBlock	Packet ingresses a blocked port	0x27	✓
StpStateLearn	Packet ingresses a port in a LEARN state	0x28	✓
IpCompMcInvalidIp	L2 compatible MC, but IP does not match	0x18	—
MyMacAndIpDisabled	Reached My-MAC over IP, but routing is disabled for InRIF	0x14	✓
MyMacAndMplsDisable	Reached My-MAC over MPLS, but MPLS is disabled for InRIF	0x15	—
ArpReply	Terminated My-MAC over ARP	0x16	—
MyMAcAndUnknownL3	Terminated My-MAC over Unknown L3	0x17	—
MplsTerminationFail	MPLS already terminated twice by label range match, additional labels may not be terminated by Label range match	0x2e	—
MplsUnexpectedBos	Terminated MPLS label indicates that another MPLS header follows, but following label is BOS	0x21	—
MplsUnexpectedNoBos	Terminated MPLS label does not indicate there's a following header, but following label is not BOS	0x22	—
L2DiscardMacsaFwd	Triggered by SA procedure (sa_drop), configured by default to FWD the packet	0x8c	—
L2DiscardMacsaDrop	Triggered by SA procedure (sa_drop), configured by default to drop the packet	0x8d	—
L2DiscardMacsaTrap	Triggered by SA procedure (sa_drop), configured by default to redirect the packet to CPU	0x8e	—
L2DiscardMacsaSnoop	Triggered by SA procedure (sa_drop), configured by default to snoop the packet	0x8f	—
L2Learn0	Triggered by SA procedure (sa_not_found), configured by default to FWD the packet	0x90	✓

Table 73: List of IRPP Predefined Traps (Continued)

bcmRxTrap…	Description	HW Trap ID	Created on Init
L2Learn1	Triggered by SA procedure (sa_not_found), configured by default to drop the packet	0x91	✓
L2Learn2	Triggered by SA procedure (sa_not_found), configured by default to redirect the packet to CPU	0x92	✓
L2Learn3	Triggered by SA procedure (sa_not_found), configured by default to snoop the packet	0x93	✓
L2DlfFwd	Triggered by Ethernet default procedure (da_not_found)	0x94	—
L2DlfDrop	Triggered by Ethernet default procedure (da_not_found)	0x95	—
L2DlfTrap	Triggered by Ethernet default procedure (da_not_found)	0x96	—
L2DlfSnoop	Triggered by Ethernet default procedure (da_not_found)	0x97	—
SameInterface	Source interface is equal to destination interface (hair-pin)	0xa3	✓
UcLooseRpfFail	Forwarding Code is IPv4 UC and RPF FEC Pointer Valid is not set	0xa6	✓
MplsUnknownLabel	In case the MPLS lookup did not hit any result	0xc	✓
Ipv6HopCount0	Terminated IPv6 header hop count (TTL) equals to zero	0x4b	—
Ipv6HopCount1	Terminated IPv6 header hop count (TTL) equals to one	0x4c	—
TcpSnFlagsZero	TCP packet, L4 sequence-number and flags are both zero	0x83	—
TcpSnZeroFlagsSet	TCP packet, L4 Sequence-Number is zero and either FIN/URG/PSH Flags are set	0x84	—
TcpSynFin	TCP packet, SYN and FIN Flags are both set	0x85	—
TcpEqualPorts	TCP packet, L4 source port equals destination port	0x86	—
TcpFragmentIncompleteHeader	TCP packet, L3 is IPv4 and IP-Header Fragmented with Fragment-Offset zero, and IP-Header (Total-Length - 4*IHL) is less than 20	0x87	—
UdpEqualPorts	UDP packet, L4 source port equals destination port	0x89	—
IcmpDataGt576	ICMP packet, L3 is IPv4 and IP-Header (Total-Length - 4*IHL) is greater than 576B or L3 is IPv6 and IP-header Payload-Length is greater than 576B	0x8a	—
IcmpFragmented	ICMP packet and IP-Header is Fragmented	0x8b	—
FailoverFacilityInvalid	Both Destination-0-Valid and Destination-1-Valid are not set	0xc0	—
UcStrictRpfFail	UC-RPF-Mode is 'Strict' and OutRIF is not equal to packet InRIF	0xc2	✓
McExplicitRpfFail	MC-RPF-Mode is 'Explicit' and RPF-Entry.Expected-InRIF is not equal to packet InRIF	0xc3	✓
McUseSipRpfFail	MC-RPF-Mode is 'Use-SIP-WITH-ECMP' and Out-RIF is not equal to In-RIF	0xc4	✓
McUseSipMultipath	MC-RPF-Mode is 'Use-SIP' and RPF-ECMP-Group-Size > 1	0xc5	—
IcmpRedirect	ICMP-Redirect is enabled, Forwarding-Code is IPv4|6-UC, and packet InRIF is equal to FEC-Entry.OutRIF	0xc6	—
OamEthAccelerated	Accelerated Ethernet OAM	0xe0	✓
FcoeSrcIdMismatchSa	FCoE packet, where FC.SrC mismatches Eth.SA	0xae	—
OamY1731MplsTp	Accelerated MPLS-TP OAM	0xe1	✓
OamY1731Pwe	Accelerated PWE OAM	0xe2	✓
OamBfdIpv4	BFD over IPv4 including single hop, multi-hop, and micro-BFD	0xe3	✓
OamBfdMpls	BFD over LSP, including IPv4 and IPv6	0xe4	✓
OamBfdPwe	BFD over PWE	0xe5	✓
OamLevel	OAM packet with an MD level below the highest MEP on active side	0xa2	✓

Table 73: List of IRPP Predefined Traps (Continued)

bcmRxTrap…	Description	HW Trap ID	Created on Init
OamPassive	OAM packet equal to or below the highest MEP level on passive side	0xac	✓
1588	IEEE 1588 protocol packet	0x98	—
Failover1Plus1Fail	One plus one protected LIF and failover status is down	0x2c	✓
DfltDroppedPacket	Default action to drop packet	0xaf	✓
DfltRedirectToCpuPacket	Default action to redirect packet to CPU	0xb3	✓
SnoopOamPacket	OAM snoop	0xfe	✓
1588Discard	IEEE 1588 protocol packet, configured by default to drop the packet	0x99	✓
1588Accepted	IEEE 1588 protocol packet, configured by default to forward the packet	0x9a	✓
EgTxFieldSnoop0	Trap code reserved for mapping to snoop command 0	0x30	✓
EgTxFieldSnoop1	Trap code reserved for mapping to snoop command 1	0x5b	✓
EgTxFieldSnoop2	Trap code reserved for mapping to snoop command 2	0x5c	✓
EgTxFieldSnoop3	Trap code reserved for mapping to snoop command 3	0x5d	✓
EgTxFieldSnoop4	Trap code reserved for mapping to snoop command 4	0x5e	✓
EgTxFieldSnoop5	Trap code reserved for mapping to snoop command 5	0x5f	✓
EgTxFieldSnoop6	Trap code reserved for mapping to snoop command 6	0x31	✓
EgTxFieldSnoop7	Trap code reserved for mapping to snoop command 7	0x61	✓
EgTxFieldSnoop8	Trap code reserved for mapping to snoop command 8	0x62	✓
EgTxFieldSnoop9	Trap code reserved for mapping to snoop command 9	0x63	✓
EgTxFieldSnoop10	Trap code reserved for mapping to snoop command 10	0x64	✓
EgTxFieldSnoop11	Trap code reserved for mapping to snoop command 11	0x65	✓
EgTxFieldSnoop12	Trap code reserved for mapping to snoop command 12	0x66	✓
EgTxFieldSnoop13	Trap code reserved for mapping to snoop command 13	0x67	✓
EgTxFieldSnoop14	Trap code reserved for mapping to snoop command 14	0xa4	✓
EgTxFieldSnoop15	Trap code reserved for mapping to snoop command 15	0xa5	✓
OamBfdIpv6	BFD over IPv6 including single hop, multi-hop, and micro-BFD	0xe8	✓
BfdOamDownMEP	Non accelerated default ingress trap for OAM and BFD	0xfd	✓
MplsTunnelTerminationTtl0	Terminated MPLS header TTL equals 0	0x23	—
MplsTunnelTerminationTtl1	Terminated MPLS header TTL equals 1	0x24	—
MplsForwardingTtl0	MPLS forwarding trap with TTL equals 0	0x81	—
MplsForwardingTtl1	MPLS forwarding trap with TTL equals 1	0x82	—
OamReflector	OAM Down MEP LBM reply through reflector	0xdf	✓
LinkLayerVlanTagDiscard	Discarded packets according to their VLAN tag configuration	0x6	✓
TerminatedVlanTagDiscard	Discarded terminated packets according to their VLAN tag configuration	0x33	✓
LinkLayerVlanTagAccept	Accepted packets according to their VLAN tag configuration	0x5	—
TerminatedVlanTagAccept	Accepted terminated packets according to their VLAN tag configuration	0x32	—
LinkLayerSaEqualsDa	Source address equals destination address	0x4	—
TerminatedSaEqualsDa	Source address equals destination address on terminated packet	0x3d	—
LinkLayerSaMulticast	Source address is multicast. The trap can be enabled per port using bcmPortControlSaMulticastEnable in `bcm_port_control_set()`.	0x3	—
TerminatedSaMulticast	Source address is multicast on terminated packet	0x3c	—

Table 73: List of IRPP Predefined Traps (Continued)

bcmRxTrap…	Description	HW Trap ID	Created on Init
LinkLayerHeaderSizeErr	Parser indicates header size error, meaning the parsed header size is greater than 144B. **NOTE:** The last parsed layer might exceed 144B, and a trap is not raised if the exceeded field is not used by the parser.	0x0	—
TerminatedHeaderSizeErr	Parser indicates terminated header size error, meaning the parsed header size is greater than 144B. **NOTE:** The last parsed layer might exceed 144B, and a trap is not raised if the exceeded field is not used by the parser.	0x39	—
ForwardingIpv4VersionError	Forwarding IPv4 header Version is different than 4	0x68	—
TerminatedIpv4VersionError	Terminated IPv4 header Version is different than 4	0x40	—
ForwardingIpv4ChecksumError	Forwarding IPv4 Internet header length is 5 and the checksum over the first 20 bytes doesn't verify	0x69	—
TerminatedIpv4ChecksumError	Terminated IPv4 Internet header length is 5 and the checksum over the first 20 bytes doesn't verify	0x41	—
ForwardingIpv4HeaderLengthError	Forwarding IPv4 Internet header length is less than 5	0x6a	—
TerminatedIpv4HeaderLengthError	Terminated IPv4 Internet header length is less than 5	0x42	—
ForwardingIpv4TotalLengthError	Forwarding IPv4 header total length is less than 20	0x6b	—
TerminatedIpv4TotalLengthError	Terminated IPv4 header total length is less than 20	0x43	—
ForwardingIpv4HasOptions	Forwarding IPv4 Internet header length is greater than 5	0x6d	—
TerminatedIpv4HasOptions	Terminated IPv4 Internet header length is greater than 5	0x45	—
ForwardingIpv4SipEqualDip	Forwarding IPv4 source IP equals to destination IP	0x6f	—
TerminatedIpv4SipEqualDip	Terminated IPv4 source IP equals to destination IP	0x47	—
ForwardingIpv4DipZero	Forwarding IPv4 destination IP equals to zero	0x70	—
TerminatedIpv4DipZero	Terminated IPv4 destination IP equals to zero	0x48	—
ForwardingIpv4SipIsMc	Forwarding IPv4 source IP is multicast	0x71	—
TerminatedIpv4SipIsMc	Terminated IPv4 source IP is multicast	0x49	—
ForwardingIpv4Ttl0	Forwarding IPv4 header TTL equals to zero	0x6c	—
TerminatedIpv4Ttl0	Terminated IPv4 header TTL equals to zero	0x44	—
ForwardingIpv4Ttl1	Forwarding IPv4 header TTL equals to one	0x6e	—
TerminatedIpv4Ttl1	Terminated IPv4 header TTL equals to one	0x46	—
ForwardingIpv6VersionError	Forwarding IPv6 header Version is different than 6	0x72	—
TerminatedIpv6VersionError	Terminated IPv6 header Version is different than 6	0x4a	—
ForwardingIpv6UnspecifiedDestination	Forwarding IPv6 header DIP is unspecified	0x75	—
TerminatedIpv6UnspecifiedDestination	Terminated IPv6 header DIP is unspecified	0x4d	—
ForwardingIpv6LoopbackAddress	Forwarding IPv6 header SIP or DIP are loopback addresses	0x76	—
TerminatedIpv6LoopbackAddress	Terminated IPv6 header SIP or DIP are loopback addresses	0x4e	—
ForwardingIpv6MulticastSource	Forwarding IPv6 header SIP is multicast	0x77	—
TerminatedIpv6MulticastSource	Terminated IPv6 header SIP is multicast	0x4f	—
ForwardingIpv6NextHeaderNull	Forwarding IPv6 header next protocol is zero	0x78	—
TerminatedIpv6NextHeaderNull	Terminated IPv6 header next protocol is zero	0x50	—
ForwardingIpv6UnspecifiedSource	Forwarding IPv6 header SIP is unspecified	0x79	—
TerminatedIpv6UnspecifiedSource	Terminated IPv6 header SIP is unspecified	0x51	—
ForwardingIpv6LocalLinkDestination	Forwarding IPv6 header DIP bits 127:118 equal to 0x3FA	0x7a	—

Table 73: List of IRPP Predefined Traps (Continued)

bcmRxTrap…	Description	HW Trap ID	Created on Init
TerminatedIpv6LocalLinkDestination	Terminated IPv6 header DIP bits 127:118 equal to 0x3FA	0x52	—
ForwardingIpv6LocalSiteDestination	Forwarding IPv6 header DIP bits 127:118 equal to 0x3FB	0x7b	—
TerminatedIpv6LocalSiteDestination	Terminated IPv6 header DIP bits 127:118 equal to 0x3FB	0x53	—
ForwardingIpv6LocalLinkSource	Forwarding IPv6 header SIP bits 127:118 equal to 0x3FA	0x7c	—
TerminatedIpv6LocalLinkSource	Terminated IPv6 header SIP bits 127:118 equal to 0x3FA	0x54	—
ForwardingIpv6LocalSiteSource	Forwarding IPv6 header SIP bits 127:118 equal to 0x3FB	0x7d	—
TerminatedIpv6LocalSiteSource	Terminated IPv6 header SIP bits 127:118 equal to 0x3FB	0x55	—
ForwardingIpv6Ipv4CompatibleDestination	Forwarding IPv6 header DIP bits 127:32 equal to zero	0x7e	—
TerminatedIpv6Ipv4CompatibleDestination	Terminated IPv6 header DIP bits 127:32 equal to zero	0x56	—
ForwardingIpv6Ipv4MappedDestination	Forwarding IPv6 header DIP bits 127:32 equal to 0000_FFFF_0000_0000_0000_0000	0x7f	—
TerminatedIpv6Ipv4MappedDestination	Terminated IPv6 header DIP bits 127:32 equal to 0000_FFFF_0000_0000_0000_0000	0x57	—
ForwardingIpv6MulticastDestination	Forwarding IPv6 header DIP is multicast	0x80	—
TerminatedIpv6MulticastDestination	Terminated IPv6 header DIP is multicast	0x58	—
TerminatedCoeFlowControl	Channelized over Ethernet flow-control terminated packet	0x37	—
TerminatedIpv4Fragmented	Terminated IPv4 header is fragmented	0x59	—
TerminatedMplsControlWordTrap	MPLS label with control word and first nibble equals one.	0x25	—
TerminatedMplsControlWordDrop	MPLS label with control word and first nibble does not equal one or zero	0x26	—
MplsPreprocessingBosOrTtl	Error of BOS or TTL was identified in MPLS preprocessing	0x1f	—
UnknownDest	Unknown destination	0xa	✓
Default	Default trap, no change to packet's forward action	0x9	✓
EgressTrapped2ndPass	Egress trapped packet on the second pass	0xb	✓
1588User1	IEEE 1588 protocol user trap 1	0x9b	—
1588User2	IEEE 1588 protocol user trap 2	0x9c	—
1588User3	IEEE 1588 protocol user trap 3	0x9d	—
1588User4	IEEE 1588 protocol user trap 4	0x9e	—
1588User5	IEEE 1588 protocol user trap 5	0x9f	—
EgBfdIpv6InvalidUdpChecksum	BFD IPv6 trap for invalid UDP checksum	0xf3	—
OamPerformanceEthAccelerated	Down MEP trapping of LM/DM to OAMP	0xa9	✓
OamPerformanceY1731MplsTp	MPLS-TP trapping of LM/DM to OAMP	0xaa	✓
OamPerformanceY1731Pwe	PWE trapping of LM/DM to OAMP	0xab	✓
TerminatedSaEqualsZero	Termination SA equals zero	0x3f	—
ForwardingSaEqualsZero	Forwarding SA equals zero	0x60	—
ForwardingIpv6Ttl0	Fwd IPv6 TTL is equal to 0	0x73	—
ForwardingIpv6Ttl1	Fwd IPv6 TTL is equal to 1	0x74	—
Srv6Usp	SRv6 USP flow with L4 header over SRv6 header	0x29	—
ItmhError	Trap for cases of ITMH Error	0x12	✓
MplsSpeculativeParseFail	MPLS speculative parsing failure	0xad	—

12.7.2.2 ERPP Application Traps

Table 74: List of ERPP Application Traps

bcmRxTrap…	Description	Created on Init
EgHairPinFilter	Source interface is equal to destination interface (hair-pin)	✓
EgSplitHorizonFilter	Triggered by conditions on incoming and outgoing packet orientation	—
EgUnknownDa	Unknown destination address	—
EgDiscardFrameTypeFilter	Unacceptable frame type on port (tagged, untagged)	—
EgIpmcTtlErr	IPv4 MC packet with invalid TTL	✓
EgIpv4Ttl0	IPv4 header TTL equals to zero	✓
EgIpv4Ttl1	IPv4 header TTL equals to one	✓
EgExcludeSrc	Trap packet of specific source	✓

12.7.2.3 ETPP Application Traps

Table 75: List of ETPP Application Traps

bcmRxTrap…	Description	Created on Init
EgTxStpStateFail	Triggered by STP state failure	✓
EgTxProtectionPathUnexpected	Unexpected traffic on a protecting path	✓
EgTxPortNotVlanMember	VID membership error	✓
EgTxDiscardFrameTypeFilter	Unacceptable frame type on port (tagged, untagged)	—
EgTxSplitHorizonFilter	Triggered by conditions on incoming and outgoing packet orientation	✓
EgTxLatency	Triggered by latency measurements	—
EgTxMetering	Triggered by meter limitations	—
EgTxGlem	Triggered when the ETPP GLEM lookup does not find an entry	—
EgTxMtuExtendedPppoe	MTU Extended PPPoE trap	—
EgTxMtuExtendedL3EgressArp	MTU Extended L3 Egress ARP trap	—

12.7.3 Trap Information Specific to the BCM88480 Device (Q2A)

12.7.3.1 Ingress Predefined Traps

Table 76: List of IRPP Predefined Traps

bcmRxTrap…	Description	HW Trap ID	Created on Init
PortNotVlanMember	Initial VID membership error	0x10	—
StpStateBlock	Packet ingresses a blocked port	0x27	✓
StpStateLearn	Packet ingresses a port in a LEARN state	0x28	✓
IpCompMcInvalidIp	L2 compatible MC, but IP does not match	0x18	—
MyMacAndIpDisabled	Reached My-MAC over IP, but routing is disabled for InRIF	0x14	✓
MyMacAndMplsDisable	Reached My-MAC over MPLS, but MPLS is disabled for InRIF	0x15	—
ArpReply	Terminated My-MAC over ARP	0x16	—
MyMAcAndUnknownL3	Terminated My-MAC over Unknown L3	0x17	—
MplsTerminationFail	MPLS already terminated twice by label range match, additional labels may not be terminated by Label range match	0x2e	—
MplsUnexpectedBos	Terminated MPLS label indicates that another MPLS header follows, but following label is BOS	0x21	—
MplsUnexpectedNoBos	Terminated MPLS label doesn't indicate there's a following header, but following label is not BOS	0x22	—
L2DiscardMacsaFwd	Triggered by SA procedure (sa_drop), configured by default to FWD the packet	0x8c	—
L2DiscardMacsaDrop	Triggered by SA procedure (sa_drop), configured by default to drop the packet	0x8d	—
L2DiscardMacsaTrap	Triggered by SA procedure (sa_drop), configured by default to redirect the packet to CPU	0x8e	—
L2DiscardMacsaSnoop	Triggered by SA procedure (sa_drop), configured by default to snoop the packet	0x8f	—
L2Learn0	Triggered by SA procedure (sa_not_found), configured by default to FWD the packet	0x90	✓
L2Learn1	Triggered by SA procedure (sa_not_found), configured by default to drop the packet	0x91	✓
L2Learn2	Triggered by SA procedure (sa_not_found), configured by default to redirect the packet to CPU	0x92	✓
L2Learn3	Triggered by SA procedure (sa_not_found), configured by default to snoop the packet	0x93	✓
L2DlfFwd	Triggered by Ethernet default procedure (da_not_found)	0x94	—
L2DlfDrop	Triggered by Ethernet default procedure (da_not_found)	0x95	—
L2DlfTrap	Triggered by Ethernet default procedure (da_not_found)	0x96	—
L2DlfSnoop	Triggered by Ethernet default procedure (da_not_found)	0x97	—
SameInterface	Source interface is equal to destination interface (hair-pin)	0xa3	✓
UcLooseRpfFail	Forwarding Code is IPv4 UC and RPF FEC Pointer Valid is not set	0xa6	✓
MplsUnknownLabel	In case the MPLS lookup did not hit any result	0xc	✓
Ipv6HopCount0	Terminated IPv6 header hop count (TTL) equals to zero	0x4b	—
Ipv6HopCount1	Terminated IPv6 header hop count (TTL) equals to one	0x4c	—
TcpSnFlagsZero	TCP packet, L4 sequence-number and flags are both zero	0x83	—

Table 76: List of IRPP Predefined Traps (Continued)

bcmRxTrap…	Description	HW Trap ID	Created on Init
TcpSnZeroFlagsSet	TCP packet, L4 Sequence-Number is zero and either FIN/URG/PSH Flags are set	0x84	—
TcpSynFin	TCP packet, SYN and FIN Flags are both set	0x85	—
TcpEqualPorts	TCP packet, L4 source port equals destination port	0x86	—
TcpFragmentIncompleteHeader	TCP packet, L3 is IPv4 and IP-Header Fragmented with Fragment-Offset zero, and IP-Header (Total-Length - 4*IHL) is less than 20	0x87	—
UdpEqualPorts	UDP packet, L4 source port equals destination port	0x89	—
IcmpDataGt576	ICMP packet, L3 is IPv4 and IP-Header (Total-Length - 4*IHL) is greater than 576B or L3 is IPv6 and IP-header Payload-Length is greater than 576B	0x8a	—
IcmpFragmented	ICMP packet and IP-Header is Fragmented	0x8b	—
FailoverFacilityInvalid	Both Destination-0-Valid and Destination-1-Valid are not set	0xc0	—
UcStrictRpfFail	UC-RPF-Mode is 'Strict' and OutRIF is not equal to packet InRIF	0xc2	✓
McExplicitRpfFail	MC-RPF-Mode is 'Explicit' and RPF-Entry.Expected-InRIF is not equal to packet InRIF	0xc3	✓
McUseSipRpfFail	MC-RPF-Mode is *Use-SIP-WITH-ECMP* and Out-RIF is not equal to In-RIF	0xc4	✓
McUseSipMultipath	MC-RPF-Mode is *Use-SIP* and RPF-ECMP-Group-Size > 1	0xc5	—
IcmpRedirect	ICMP-Redirect is enabled, Forwarding-Code is IPv4|6-UC, and packet InRIF is equal to FEC-Entry.OutRIF	0xc6	—
OamEthAccelerated	Accelerated Ethernet OAM	0xe0	✓
FcoeSrcIdMismatchSa	FCoE packet, where FC.SrC mismatches Eth.SA	0xae	—
OamY1731MplsTp	Accelerated MPLS-TP OAM	0xe1	✓
OamY1731Pwe	Accelerated PWE OAM	0xe2	✓
OamBfdIpv4	BFD over IPv4 including single hop, multi-hop, and micro-BFD	0xe3	✓
OamBfdMpls	BFD over LSP, including IPv4 and IPv6	0xe4	✓
OamBfdPwe	BFD over PWE	0xe5	✓
OamLevel	OAM packet with an MD level below the highest MEP on active side	0xa2	✓
OamPassive	OAM packet equal to or below the highest MEP level on passive side	0xac	✓
1588	IEEE 1588 protocol packet	0x98	—
Failover1Plus1Fail	One plus one protected LIF and failover status is down	0x2c	✓
DfltDroppedPacket	Default action to drop packet	0xaf	✓
DfltRedirectToCpuPacket	Default action to redirect packet to CPU	0xb3	✓
SnoopOamPacket	OAM snoop	0xfe	✓
1588Discard	IEEE 1588 protocol packet, configured by default to drop the packet	0x99	✓
1588Accepted	IEEE 1588 protocol packet, configured by default to forward the packet	0x9a	—
EgTxFieldSnoop0	Trap code reserved for mapping to snoop command 0	0x30	✓
EgTxFieldSnoop1	Trap code reserved for mapping to snoop command 1	0x5b	✓
EgTxFieldSnoop2	Trap code reserved for mapping to snoop command 2	0x5c	✓
EgTxFieldSnoop3	Trap code reserved for mapping to snoop command 3	0x5d	✓
EgTxFieldSnoop4	Trap code reserved for mapping to snoop command 4	0x5e	✓
EgTxFieldSnoop5	Trap code reserved for mapping to snoop command 5	0x5f	✓
EgTxFieldSnoop6	Trap code reserved for mapping to snoop command 6	0x31	✓
EgTxFieldSnoop7	Trap code reserved for mapping to snoop command 7	0x61	✓

Table 76: List of IRPP Predefined Traps (Continued)

bcmRxTrap…	Description	HW Trap ID	Created on Init
EgTxFieldSnoop8	Trap code reserved for mapping to snoop command 8	0x62	✓
EgTxFieldSnoop9	Trap code reserved for mapping to snoop command 9	0x63	✓
EgTxFieldSnoop10	Trap code reserved for mapping to snoop command 10	0x64	✓
EgTxFieldSnoop11	Trap code reserved for mapping to snoop command 11	0x65	✓
EgTxFieldSnoop12	Trap code reserved for mapping to snoop command 12	0x66	✓
EgTxFieldSnoop13	Trap code reserved for mapping to snoop command 13	0x67	✓
EgTxFieldSnoop14	Trap code reserved for mapping to snoop command 14	0xa4	✓
EgTxFieldSnoop15	Trap code reserved for mapping to snoop command 15	0xa5	✓
OamBfdIpv6	BFD over IPv6 including single hop, multi-hop, and micro-BFD	0xe8	✓
BfdOamDownMEP	Non-accelerated default ingress trap for OAM and BFD	0xfd	✓
MplsTunnelTerminationTtl0	Terminated IPv4 header TTL equals 0	0x23	—
MplsTunnelTerminationTtl1	Terminated IPv4 header TTL equals 1	0x24	—
MplsForwardingTtl0	MPLS forwarding trap with TTL equals 0	0x81	—
MplsForwardingTtl1	MPLS forwarding trap with TTL equals 1	0x82	—
OamReflector	OAM Down MEP LBM reply through reflector	0xdf	✓
LinkLayerVlanTagDiscard	Discarded packets according to their VLAN tag configuration	0x6	✓
TerminatedVlanTagDiscard	Discarded terminated packets according to their VLAN tag configuration	0x33	✓
LinkLayerVlanTagAccept	Accepted packets according to their VLAN tag configuration	0x5	—
TerminatedVlanTagAccept	Accepted terminated packets according to their VLAN tag configuration	0x32	—
LinkLayerSaEqualsDa	Source address equals destination address	0x4	—
TerminatedSaEqualsDa	Source address equals destination address on terminated packet	0x3d	—
LinkLayerSaMulticast	Source address is multicast, the trap can be enabled per port using bcmPortControlSaMulticastEnable in `bcm_port_control_set()`	0x3	—
TerminatedSaMulticast	Source address is multicast on terminated packet	0x3c	—
LinkLayerHeaderSizeErr	Parser indicates header size error, meaning the parsed header size is greater than 144B. **NOTE:** The last parsed layer might exceed 144B, and a trap is not raised if the exceeded field is not used by the parser.	0x0	—
TerminatedHeaderSizeErr	Parser indicates terminated header size error, meaning the parsed header size is greater than 144B. **NOTE:** The last parsed layer might exceed 144B, and a trap is not raised if the exceeded field is not used by the parser.	0x39	—
ForwardingIpv4VersionError	Forwarding IPv4 header Version is different than 4	0x68	—
TerminatedIpv4VersionError	Terminated IPv4 header Version is different than 4	0x40	—
ForwardingIpv4ChecksumError	Forwarding IPv4 Internet header length is 5 and the checksum over the first 20 bytes doesn't verify	0x69	—
TerminatedIpv4ChecksumError	Terminated IPv4 Internet header length is 5 and the checksum over the first 20 bytes doesn't verify	0x41	—
ForwardingIpv4HeaderLengthError	Forwarding IPv4 Internet header length is less than 5	0x6a	—
TerminatedIpv4HeaderLengthError	Terminated IPv4 Internet header length is less than 5	0x42	—
ForwardingIpv4TotalLengthError	Forwarding IPv4 header total length is less than 20	0x6b	—
TerminatedIpv4TotalLengthError	Terminated IPv4 header total length is less than 20	0x43	—

Table 76: List of IRPP Predefined Traps (Continued)

bcmRxTrap…	Description	HW Trap ID	Created on Init
ForwardingIpv4HasOptions	Forwarding IPv4 Internet header length is greater than 5	0x6d	—
TerminatedIpv4HasOptions	Terminated IPv4 Internet header length is greater than 5	0x45	—
ForwardingIpv4SipEqualDip	Forwarding IPv4 source IP equals to destination IP	0x6f	—
TerminatedIpv4SipEqualDip	Terminated IPv4 source IP equals to destination IP	0x47	—
ForwardingIpv4DipZero	Forwarding IPv4 destination IP equals to zero	0x70	—
TerminatedIpv4DipZero	Terminated IPv4 destination IP equals to zero	0x48	—
ForwardingIpv4SipIsMc	Forwarding IPv4 source IP is multicast	0x71	—
TerminatedIpv4SipIsMc	Terminated IPv4 source IP is multicast	0x49	—
ForwardingIpv4Ttl0	Forwarding IPv4 header TTL equals to zero	0x6c	—
TerminatedIpv4Ttl0	Terminated IPv4 header TTL equals to zero	0x44	—
ForwardingIpv4Ttl1	Forwarding IPv4 header TTL equals to one	0x6e	—
TerminatedIpv4Ttl1	Terminated IPv4 header TTL equals to one	0x46	—
ForwardingIpv6VersionError	Forwarding IPv6 header Version is different than 6	0x72	—
TerminatedIpv6VersionError	Terminated IPv6 header Version is different than 6	0x4a	—
ForwardingIpv6UnspecifiedDestination	Forwarding IPv6 header DIP is unspecified	0x75	—
TerminatedIpv6UnspecifiedDestination	Terminated IPv6 header DIP is unspecified	0x4d	—
ForwardingIpv6LoopbackAddress	Forwarding IPv6 header SIP or DIP are loopback addresses	0x76	—
TerminatedIpv6LoopbackAddress	Terminated IPv6 header SIP or DIP are loopback addresses	0x4e	—
ForwardingIpv6MulticastSource	Forwarding IPv6 header SIP is multicast	0x77	—
TerminatedIpv6MulticastSource	Terminated IPv6 header SIP is multicast	0x4f	—
ForwardingIpv6NextHeaderNull	Forwarding IPv6 header next protocol is zero	0x78	—
TerminatedIpv6NextHeaderNull	Terminated IPv6 header next protocol is zero	0x50	—
ForwardingIpv6UnspecifiedSource	Forwarding IPv6 header SIP is unspecified	0x79	—
TerminatedIpv6UnspecifiedSource	Terminated IPv6 header SIP is unspecified	0x51	—
ForwardingIpv6LocalLinkDestination	Forwarding IPv6 header DIP bits 127:118 equal to 0x3FA	0x7a	—
TerminatedIpv6LocalLinkDestination	Terminated IPv6 header DIP bits 127:118 equal to 0x3FA	0x52	—
ForwardingIpv6LocalSiteDestination	Forwarding IPv6 header DIP bits 127:118 equal to 0x3FB	0x7b	—
TerminatedIpv6LocalSiteDestination	Terminated IPv6 header DIP bits 127:118 equal to 0x3FB	0x53	—
ForwardingIpv6LocalLinkSource	Forwarding IPv6 header SIP bits 127:118 equal to 0x3FA	0x7c	—
TerminatedIpv6LocalLinkSource	Terminated IPv6 header SIP bits 127:118 equal to 0x3FA	0x54	—
ForwardingIpv6LocalSiteSource	Forwarding IPv6 header SIP bits 127:118 equal to 0x3FB	0x7d	—
TerminatedIpv6LocalSiteSource	Terminated IPv6 header SIP bits 127:118 equal to 0x3FB	0x55	—
ForwardingIpv6Ipv4CompatibleDestination	Forwarding IPv6 header DIP bits 127:32 equal to zero	0x7e	—
TerminatedIpv6Ipv4CompatibleDestination	Terminated IPv6 header DIP bits 127:32 equal to zero	0x56	—
ForwardingIpv6Ipv4MappedDestination	Forwarding IPv6 header DIP bits 127:32 equal to 0000_FFFF_0000_0000_0000_0000	0x7f	—
TerminatedIpv6Ipv4MappedDestination	Terminated IPv6 header DIP bits 127:32 equal to 0000_FFFF_0000_0000_0000_0000	0x57	—
ForwardingIpv6MulticastDestination	Forwarding IPv6 header DIP is multicast	0x80	—
TerminatedIpv6MulticastDestination	Terminated IPv6 header DIP is multicast	0x58	—
TerminatedCoeFlowControl	Channelized over Ethernet flow-control terminated packet	0x37	—
TerminatedIpv4Fragmented	Terminated IPv4 header is fragmented	0x59	—

Table 76: List of IRPP Predefined Traps (Continued)

bcmRxTrap…	Description	HW Trap ID	Created on Init
TerminatedMplsControlWordTrap	MPLS label with control word and first nibble equals one.	0x25	—
TerminatedMplsControlWordDrop	MPLS label with control word and first nibble does not equal one or zero	0x26	—
MplsPreprocessingBosOrTtl	Error of BOS or TTL was identified in MPLS preprocessing	0x1f	—
UnknownDest	Unknown destination	0xa	✓
Default	Default trap, no change to packet's forward action	0x9	✓
EgressTrapped2ndPass	Egress trapped packet on 2nd pass	0xb	✓
1588User1	IEEE 1588 protocol user trap 1	0x9b	—
1588User2	IEEE 1588 protocol user trap 2	0x9c	—
1588User3	IEEE 1588 protocol user trap 3	0x9d	—
1588User4	IEEE 1588 protocol user trap 4	0x9e	—
1588User5	IEEE 1588 protocol user trap 5	0x9f	—
EgBfdIpv6InvalidUdpChecksum	BFD IPv6 trap for invalid UDP checksum	0xf3	—
OamPerformanceEthAccelerated	Down MEP trapping of LM/DM to OAMP	0xa9	✓
OamPerformanceY1731MplsTp	MPLS-TP trapping of LM/DM to OAMP	0xaa	✓
OamPerformanceY1731Pwe	PWE trapping of LM/DM to OAMP	0xab	✓
TerminatedSaEqualsZero	Termination SA equals zero	0x3f	—
ForwardingSaEqualsZero	Forwarding SA equals zero	0x60	—
ForwardingIpv6Ttl0	Fwd IPv6 TTL is equal to 0	0x73	—
ForwardingIpv6Ttl1	Fwd IPv6 TTL is equal to 1	0x74	—
Srv6Usp	SRv6 USP flow with L4 header over SRv6 header	0x29	—
ItmhError	Trap for cases of ITMH Error	0x12	✓
MplsSpeculativeParseFail	MPLS speculative parsing failure	0xad	—

12.7.3.2 ERPP Application Traps

Table 77: List of ERPP Application Traps

bcmRxTrap…	Description	Created on Init
EgHairPinFilter	Source interface is equal to destination interface (hair-pin)	✓
EgSplitHorizonFilter	Triggered by conditions on incoming and outgoing packet orientation	—
EgUnknownDa	Unknown destination address	—
EgDiscardFrameTypeFilter	Unacceptable frame type on port (tagged, untagged)	—
EgIpmcTtlErr	IPv4 MC packet with invalid TTL	✓
EgIpv4Ttl0	IPv4 header TTL equals to zero	✓
EgIpv4Ttl1	IPv4 header TTL equals to one	✓
EgExcludeSrc	Trap packet of specific source	✓

12.7.3.3 ETPP Application Traps

Table 78: List of ETPP Application Traps

bcmRxTrap…	Description	Created on Init
EgTxStpStateFail	Triggered by STP state failure	✓
EgTxProtectionPathUnexpected	Unexpected traffic on a protecting path	✓
EgTxPortNotVlanMember	VID membership error	✓
EgTxDiscardFrameTypeFilter	Unacceptable frame type on port (tagged, untagged)	—
EgTxSplitHorizonFilter	Triggered by conditions on incoming and outgoing packet orientation	✓
EgTxLatency	Triggered by latency measurements	—
EgTxMetering	Triggered by meter limitations	—
EgTxGlem	Triggered when the ETPP GLEM lookup does not find an entry	—
EgTxMtuExtendedPppoe	MTU Extended PPPoE trap	—
EgTxMtuExtendedL3EgressArp	MTU Extended L3 Egress ARP trap	—

12.7.4 Trap Information Specific to the BCM88830 Device (J2X)

12.7.4.1 Ingress Predefined Traps

Table 79: List of IRPP Predefined Traps

bcmRxTrap…	Description	HW Trap ID	Created on Init
PortNotVlanMember	Initial VID membership error	0x10	—
StpStateBlock	Packet ingresses a blocked port	0x27	✓
StpStateLearn	Packet ingresses a port in a LEARN state	0x28	✓
IpCompMcInvalidIp	L2 compatible MC, but IP does not match	0x18	—
MyMacAndIpDisabled	Reached My-MAC over IP, but routing is disabled for InRIF	0x14	✓
MyMacAndMplsDisable	Reached My-MAC over MPLS, but MPLS is disabled for InRIF	0x15	—
ArpReply	Terminated My-MAC over ARP	0x16	—
MyMAcAndUnknownL3	Terminated My-MAC over Unknown L3	0x17	—
MplsTerminationFail	MPLS already terminated twice by label range match, additional labels may not be terminated by Label range match	0x2e	—
MplsUnexpectedBos	Terminated MPLS label indicates that another MPLS header follows, but following label is BOS	0x21	—
MplsUnexpectedNoBos	Terminated MPLS label doesn't indicate there's a following header, but following label is not BOS	0x22	—
L2DiscardMacsaFwd	Triggered by SA procedure (sa_drop), configured by default to FWD the packet	0x8c	—
L2DiscardMacsaDrop	Triggered by SA procedure (sa_drop), configured by default to Drop the packet	0x8d	—
L2DiscardMacsaTrap	Triggered by SA procedure (sa_drop), configured by default to redirect the packet to CPU	0x8e	—
L2DiscardMacsaSnoop	Triggered by SA procedure (sa_drop), configured by default to Snoop the packet	0x8f	—
L2Learn0	Triggered by SA procedure (sa_not_found), configured by default to FWD the packet	0x90	✓

Table 79: List of IRPP Predefined Traps

bcmRxTrap…	Description	HW Trap ID	Created on Init
L2Learn1	Triggered by SA procedure (sa_not_found), configured by default to Drop the packet	0x91	✓
L2Learn2	Triggered by SA procedure (sa_not_found), configured by default to redirect the packet to CPU	0x92	✓
L2Learn3	Triggered by SA procedure (sa_not_found), configured by default to Snoop the packet	0x93	✓
ExternalLookupError	ELK is accessed and returns an error	0xa0	—
L2DlfFwd	Triggered by Ethernet default procedure (da_not_found)	0x94	—
L2DlfDrop	Triggered by Ethernet default procedure (da_not_found)	0x95	—
L2DlfTrap	Triggered by Ethernet default procedure (da_not_found)	0x96	—
L2DlfSnoop	Triggered by Ethernet default procedure (da_not_found)	0x97	—
SameInterface	Source interface is equal to destination interface (hair-pin)	0xa3	✓
UcLooseRpfFail	Forwarding Code is IPv4 UC and RPF FEC Pointer Valid is not set	0xa6	✓
MplsUnknownLabel	In case the MPLS lookup didnt hit any result	0xc	✓
Ipv6HopCount0	Terminated IPv6 header Hop count (TTL) equals to zero	0x4b	—
Ipv6HopCount1	Terminated IPv6 header Hop count (TTL) equals to one	0x4c	—
TcpSnFlagsZero	TCP packet, L4 Sequence-Number and Flags are both zero	0x83	—
TcpSnZeroFlagsSet	TCP packet, L4 Sequence-Number is zero and either FIN/URG/PSH Flags are set	0x84	—
TcpSynFin	TCP packet, SYN and FIN Flags are both set	0x85	—
TcpEqualPorts	TCP packet, L4 source port equals destination port	0x86	—
TcpFragmentIncompleteHeader	TCP packet, L3 is IPv4 and IP-Header Fragmented with Fragment-Offset zero, and IP-Header (Total-Length - 4*IHL) is less than 20	0x87	—
UdpEqualPorts	UDP packet, L4 source port equals destination port	0x89	—
IcmpDataGt576	ICMP packet, L3 is IPv4 and IP-Header (Total-Length - 4*IHL) is greater than 576B or L3 is IPv6 and IP-header Payload-Length is greater than 576B	0x8a	—
IcmpFragmented	ICMP packet and IP-Header is Fragmented	0x8b	—
FailoverFacilityInvalid	Both Destination-0-Valid and Destination-1-Valid are not set	0xc0	—
UcStrictRpfFail	UC-RPF-Mode is 'Strict' and OutRIF is not equal to packet InRIF	0xc2	✓
McExplicitRpfFail	MC-RPF-Mode is 'Explicit' and RPF-Entry.Expected-InRIF is not equal to packet InRIF	0xc3	✓
McUseSipRpfFail	MC-RPF-Mode is 'Use-SIP-WITH-ECMP' and Out-RIF is not equal to In-RIF	0xc4	✓
McUseSipMultipath	MC-RPF-Mode is 'Use-SIP' and RPF-ECMP-Group-Size > 1	0xc5	—
IcmpRedirect	ICMP-Redirect is enabled, Forwarding-Code is IPv4|6-UC, and packet InRIF is equal to FEC-Entry.OutRIF	0xc6	—
OamEthAccelerated	Accelerated Ethernet OAM	0xe0	✓
FcoeSrcIdMismatchSa	FCoE packet, where FC.SrC mismatchs Eth.SA	0xae	—
OamY1731MplsTp	Accelerated MPLS-TP OAM	0xe1	✓
OamY1731Pwe	Accelerated PWE OAM	0xe2	✓
OamBfdIpv4	BFD over IPv4 including single hop, multi-hop and micro-BFD	0xe3	✓
OamBfdMpls	BFD over LSP, including IPv4 and IPv6	0xe4	✓
OamBfdPwe	BFD over PWE	0xe5	✓

Table 79: List of IRPP Predefined Traps

bcmRxTrap…	Description	HW Trap ID	Created on Init
OamLevel	OAM packet with mdlevel below highest MEP on active side	0xa2	✓
OamPassive	OAM packet equal or below highest MEP level on passive side	0xac	✓
1588	1588 protocol packet	0x98	—
Failover1Plus1Fail	One plus one protected LIF and failover status is down	0x2c	✓
DfltDroppedPacket	Deafult action to drop packet	0xaf	✓
DfltRedirectToCpuPacket	Deafult action to redirect packet to CPU	0xb3	✓
SnoopOamPacket	OAM snoop	0xfe	✓
1588Discard	1588 protocol packet, configured by default to Drop the packet	0x99	✓
1588Accepted	1588 protocol packet, configured by default to FWD the packet	0x9a	—
EgTxFieldSnoop0	Trap code reserved for mapping to snoop command 0	0x30	✓
EgTxFieldSnoop1	Trap code reserved for mapping to snoop command 1	0x5b	✓
EgTxFieldSnoop2	Trap code reserved for mapping to snoop command 2	0x5c	✓
EgTxFieldSnoop3	Trap code reserved for mapping to snoop command 3	0x5d	✓
EgTxFieldSnoop4	Trap code reserved for mapping to snoop command 4	0x5e	✓
EgTxFieldSnoop5	Trap code reserved for mapping to snoop command 5	0x5f	✓
EgTxFieldSnoop6	Trap code reserved for mapping to snoop command 6	0x31	✓
EgTxFieldSnoop7	Trap code reserved for mapping to snoop command 7	0x61	✓
EgTxFieldSnoop8	Trap code reserved for mapping to snoop command 8	0x62	✓
EgTxFieldSnoop9	Trap code reserved for mapping to snoop command 9	0x63	✓
EgTxFieldSnoop10	Trap code reserved for mapping to snoop command 10	0x64	✓
EgTxFieldSnoop11	Trap code reserved for mapping to snoop command 11	0x65	✓
EgTxFieldSnoop12	Trap code reserved for mapping to snoop command 12	0x66	✓
EgTxFieldSnoop13	Trap code reserved for mapping to snoop command 13	0x67	✓
EgTxFieldSnoop14	Trap code reserved for mapping to snoop command 14	0xa4	✓
EgTxFieldSnoop15	Trap code reserved for mapping to snoop command 15	0xa5	✓
OamBfdIpv6	BFD over IPv6 including single hop, multi-hop and micro-BFD	0xe8	✓
BfdOamDownMEP	Non accelerated default ingress trap for OAM and BFD	0xfd	✓
MplsTunnelTerminationTtl0	Terminated IPv4 header TTL equals to zero	0x23	—
MplsTunnelTerminationTtl1	Terminated IPv4 header TTL equals to one	0x24	—
MplsForwardingTtl0	MPLS forwarding trap with TTL equal to 0	0x81	—
MplsForwardingTtl1	MPLS forwarding trap with TTL equal to 1	0x82	—
OamReflector	OAM Down MEP LBM reply through reflector	0xdf	✓
LinkLayerVlanTagDiscard	Discarded packets according to their VLAN Tag configuration	0x6	✓
TerminatedVlanTagDiscard	Discarded terminated packets according to their VLAN Tag configuration	0x33	✓
LinkLayerVlanTagAccept	Accepted packets according to their VLAN Tag configuration	0x5	—
TerminatedVlanTagAccept	Accepted terminated packets according to their VLAN Tag configuration	0x32	—
LinkLayerSaEqualsDa	Source address equals destination address	0x4	—
TerminatedSaEqualsDa	Source address equals destination address on terminated packet	0x3d	—
LinkLayerSaMulticast	Source address is multicast, the trap can be enabled per port using bcmPortControlSaMulticastEnable in bcm_port_control_set()	0x3	—
TerminatedSaMulticast	Source address is multicast on terminated packet	0x3c	—

Table 79: List of IRPP Predefined Traps

bcmRxTrap…	Description	HW Trap ID	Created on Init
LinkLayerHeaderSizeErr	Parser indicates header size error, meaning Parsed Header size is > 144B. Note that last parsed layer might exceed 144B and trap will not be raised if the exceeded field are not used by Parser	0x0	—
TerminatedHeaderSizeErr	Parser indicates terminated header size error, meaning Parsed Header size is > 144B. Note that last parsed layer might exceed 144B and trap will not be raised if the exceeded field are not used by Parser	0x39	—
ForwardingIpv4VersionError	Forwarding IPv4 header Version is different than 4	0x68	—
TerminatedIpv4VersionError	Terminated IPv4 header Version is different than 4	0x40	—
ForwardingIpv4ChecksumError	Forwarding IPv4 Internet Header Length is 5 and the checksum over the first 20 bytes doesn't verify	0x69	—
TerminatedIpv4ChecksumError	Terminated IPv4 Internet Header Length is 5 and the checksum over the first 20 bytes doesn't verify	0x41	—
ForwardingIpv4HeaderLengthError	Forwarding IPv4 Internet Header Length is less than 5	0x6a	—
TerminatedIpv4HeaderLengthError	Terminated IPv4 Internet Header Length is less than 5	0x42	—
ForwardingIpv4TotalLengthError	Forwarding IPv4 header Total length is less than 20	0x6b	—
TerminatedIpv4TotalLengthError	Terminated IPv4 header Total length is less than 20	0x43	—
ForwardingIpv4HasOptions	Forwarding IPv4 Internet Header Length is greater than 5	0x6d	—
TerminatedIpv4HasOptions	Terminated IPv4 Internet Header Length is greater than 5	0x45	—
ForwardingIpv4SipEqualDip	Forwarding IPv4 source IP equals to destination IP	0x6f	—
TerminatedIpv4SipEqualDip	Terminated IPv4 source IP equals to destination IP	0x47	—
ForwardingIpv4DipZero	Forwarding IPv4 destination IP equals to zero	0x70	—
TerminatedIpv4DipZero	Terminated IPv4 destination IP equals to zero	0x48	—
ForwardingIpv4SipIsMc	Forwarding IPv4 source ip is multicast	0x71	—
TerminatedIpv4SipIsMc	Terminated IPv4 source ip is multicast	0x49	—
ForwardingIpv4Ttl0	Forwarding IPv4 header TTL equals to zero	0x6c	—
TerminatedIpv4Ttl0	Terminated IPv4 header TTL equals to zero	0x44	—
ForwardingIpv4Ttl1	Forwarding IPv4 header TTL equals to one	0x6e	—
TerminatedIpv4Ttl1	Terminated IPv4 header TTL equals to one	0x46	—
ForwardingIpv6VersionError	Forwarding IPv6 header Version is different than 6	0x72	—
TerminatedIpv6VersionError	Terminated IPv6 header Version is different than 6	0x4a	—
ForwardingIpv6UnspecifiedDestination	Forwarding IPv6 header DIP is unspecified	0x75	—
TerminatedIpv6UnspecifiedDestination	Terminated IPv6 header DIP is unspecified	0x4d	—
ForwardingIpv6LoopbackAddress	Forwarding IPv6 header SIP or DIP are loopback addresses	0x76	—
TerminatedIpv6LoopbackAddress	Terminated IPv6 header SIP or DIP are loopback addresses	0x4e	—
ForwardingIpv6MulticastSource	Forwarding IPv6 header SIP is Multicast	0x77	—
TerminatedIpv6MulticastSource	Terminated IPv6 header SIP is Multicast	0x4f	—
ForwardingIpv6NextHeaderNull	Forwarding IPv6 header next protocol is zero	0x78	—
TerminatedIpv6NextHeaderNull	Terminated IPv6 header next protocol is zero	0x50	—
ForwardingIpv6UnspecifiedSource	Forwarding IPv6 header SIP is unspecified	0x79	—
TerminatedIpv6UnspecifiedSource	Terminated IPv6 header SIP is unspecified	0x51	—
ForwardingIpv6LocalLinkDestination	Forwarding IPv6 header DIP bits 127:118 equal to 0x3FA	0x7a	—
TerminatedIpv6LocalLinkDestination	Terminated IPv6 header DIP bits 127:118 equal to 0x3FA	0x52	—
ForwardingIpv6LocalSiteDestination	Forwarding IPv6 header DIP bits 127:118 equal to 0x3FB	0x7b	—

Table 79: List of IRPP Predefined Traps

bcmRxTrap…	Description	HW Trap ID	Created on Init
TerminatedIpv6LocalSiteDestination	Terminated IPv6 header DIP bits 127:118 equal to 0x3FB	0x53	—
ForwardingIpv6LocalLinkSource	Forwarding IPv6 header SIP bits 127:118 equal to 0x3FA	0x7c	—
TerminatedIpv6LocalLinkSource	Terminated IPv6 header SIP bits 127:118 equal to 0x3FA	0x54	—
ForwardingIpv6LocalSiteSource	Forwarding IPv6 header SIP bits 127:118 equal to 0x3FB	0x7d	—
TerminatedIpv6LocalSiteSource	Terminated IPv6 header SIP bits 127:118 equal to 0x3FB	0x55	—
ForwardingIpv6Ipv4CompatibleDestination	Forwarding IPv6 header DIP bits 127:32 equal to zero	0x7e	—
TerminatedIpv6Ipv4CompatibleDestination	Terminated IPv6 header DIP bits 127:32 equal to zero	0x56	—
ForwardingIpv6Ipv4MappedDestination	Forwarding IPv6 header DIP bits 127:32 equal to 0000_FFFF_0000_0000_0000_0000	0x7f	—
TerminatedIpv6Ipv4MappedDestination	Terminated IPv6 header DIP bits 127:32 equal to 0000_FFFF_0000_0000_0000_0000	0x57	—
ForwardingIpv6MulticastDestination	Forwarding IPv6 header DIP is Multicast	0x80	—
TerminatedIpv6MulticastDestination	Terminated IPv6 header DIP is Multicast	0x58	—
TerminatedCoeFlowControl	Channelized over Ethernet Flow Control terminated Packet	0x37	—
TerminatedIpv4Fragmented	Terminated IPv4 header is fragmented	0x59	—
TerminatedMplsControlWordTrap	MPLS label with control word and first nibble equals one	0x25	—
TerminatedMplsControlWordDrop	MPLS label with control word and first nibble doesn't equal one or zero	0x26	—
MplsPreprocessingBosOrTtl	Error of BOS or TTL was identified in MPLS preprocessing	0x1f	—
UnknownDest	Unknown destination	0xa	—
Default	Default trap, no change to packet's forward action	0x9	—
EgressTrapped2ndPass	Egress trapped packet on 2nd pass	0xb	—
1588User1	1588 protocol user trap 1	0x9b	—
1588User2	1588 protocol user trap 2	0x9c	—
1588User3	1588 protocol user trap 3	0x9d	—
1588User4	1588 protocol user trap 4	0x9e	—
1588User5	1588 protocol user trap 5	0x9f	—
EgBfdIpv6InvalidUdpChecksum	BFD IPv6 trap for invalid UDP checksum	0xf3	—
OamPerformanceEthAccelerated	Down MEP trapping of LM/DM to OAMP	0xa9	✓
OamPerformanceY1731MplsTp	MPLS-TP trapping of LM/DM to OAMP	0xaa	✓
OamPerformanceY1731Pwe	PWE trapping of LM/DM to OAMP	0xab	✓
TerminatedSaEqualsZero	Termination SA equals Zero	0x3f	—
ForwardingSaEqualsZero	Forwarding SA equals Zero	0x60	—
ForwardingIpv6Ttl0	Fwd IPv6 TTL is equal to 0	0x73	—
ForwardingIpv6Ttl1	Fwd IPv6 TTL is equal to 1	0x74	—
Srv6Usp	SRv6 USP flow with L4 header over SRv6 header	0x29	—
ItmhError	Trap for cases of ITMH Error	0x12	✓
MplsSpeculativeParseFail	MPLS speculative parsing failure	0xad	—

12.7.4.2 ERPP Application Traps

Table 80: List of ERPP Application Traps

bcmRxTrap…	Description	Created on Init
EgHairPinFilter	Source interface is equal to destination interface (hair-pin)	✓
EgSplitHorizonFilter	Triggered by conditions on incoming and outgoing packet orientation	✓
EgUnknownDa	Unknown Destination address	—
EgDiscardFrameTypeFilter	Unacceptable frame type on port (tagged, untagged)	—
EgIpmcTtlErr	IPv4 MC packet with invalid TTL	✓
EgIpv4VersionError	Egress trap: Version is different than 4.	✓
EgIpv4ChecksumError	Egress trap: IHL is 5 and the checksum over the first 20 bytes does not verify.	✓
EgIpv4HeaderLengthError	Egress trap: IHL (Internet Header Length) is less than 5.	✓
EgIpv4TotalLengthError	Egress trap: Total length is less than 20.	✓
EgIpv4Ttl0	IPv4 header TTL equals to zero	✓
EgIpv4HasOptions	Egress trap: IHL (Internet Header Length) is greater than 5.	✓
EgIpv4Ttl1	IPv4 header TTL equals to one	✓
EgIpv4SipEqualDip	Egress trap: Source-IP is equal to destination IP.	✓
EgIpv4DipZero	Egress trap: Destination IP is 0.	✓
EgIpv4SipIsMc	Egress trap: Source-IP is multicast.	✓
EgIpv6UnspecifiedDestination	Egress trap: Forwarding header DIP = ::.	✓
EgIpv6LoopbackAddress	Egress trap: DIP = ::1 or SIP = ::1.	✓
EgIpv6MulticastSource	Egress trap: The MSB of the SIP = 0xFF.	✓
EgIpv6UnspecifiedSource	Egress trap: SIP = ::.	✓
EgIpv6LocalLinkDestination	Egress trap: Bits 127:118 of the destination-IP are equal to 0x3FA.	✓
EgIpv6LocalSiteDestination	Egress trap: Bits 127:118 of the DIP = 0x3FB (deprecated).	✓
EgIpv6LocalLinkSource	Egress trap: Bits 127:118 of the SIP = 0x3FA.	✓
EgIpv6LocalSiteSource	Egress trap: Bits 127:118 of the SIP = 0x3FB.	✓
EgIpv6Ipv4CompatibleDestination	Egress trap: Bits 127:32 of the DIP = 0.	✓
EgIpv6Ipv4MappedDestination	Egress trap: Bits 127:32 of the DIP are equal to 0000_0000_0000_0000_0000_FFFF.	✓
EgIpv6MulticastDestination	Egress trap: MSB of the DIP=0xFF.	—
EgIpv6NextHeaderNull	Egress trap: Next-protocol is zero.	✓
EgExcludeSrc	Trap packet of specific source	—
EgTcpSnFlagsZero	ERPP L4 Sequence-Number and Flags (6) are both zero.	✓
EgTcpSnZeroFlagsSet	ERPP L4 Sequence-Number is zero and either Flags. FIN, Flags. URG or FLAGS. PSH are set.	✓
EgTcpSynFin	ERPP Both Flags. SYN and Flags. FIN are set.	✓
EgTcpEqualPorts	ERPP Source-Port equals Destination-Port.	✓
EgTcpFragmentIncompleteHeader	ERPP L3 is IPv4 and IP-Header. Fragmented and IP-Header. Fragment-Offset zero and (IPv4-Header. Total-Length - 4 * IPv4-Header. IHL) is less than 20B.	✓
EgTcpFragmentOffsetLt8	ERPP L3 is IPv4 and IP-Header. Fragmented and IP-Header. Fragment-Offset is less than 8.	✓
EgUdpEqualPorts	ERPP Source-Port equals Destination-Port.	✓
EgIpv6VersionError	ERPP Version is different than 6.	✓

12.7.4.3 ETPP Application Traps

Table 81: ETPP Application Traps

bcmRxTrap…	Description	Created on Init
EgTxStpStateFail	Triggered by STP state failure	✓
EgTxProtectionPathUnexpected	Unexpected Traffic on a Protected Path	✓
EgTxPortNotVlanMember	VID membership error	✓
EgTxDiscardFrameTypeFilter	Unacceptable frame type on port (tagged, untagged)	—
EgTxSplitHorizonFilter	Triggered by conditions on incoming and outgoing packet orientation	✓
EgTxLatency	Triggered by latency measurements	—
EgTxMetering	Triggered by meter limitations	—
EgTxGlem	Triggered when ETPP GLEM lookup does not find an entry	—
EgTxMtuExtendedPppoe	MTU Extended PPPoE trap	—
EgTxMtuExtendedL3EgressArp	MTU Extended L3 Egress ARP trap	—

Chapter 13: Load Balancing

13.1 Introduction

This chapter describes the basic process of generating load-balancing keys without the advanced options of the ACL (Ingress PMF block), which can modify or overwrite these key values. For details about advanced ACL options, see Chapter 11, Field Processor.

The device creates five load-balancing keys for the five load balancing clients. Three keys are used for ECMP (hierarchies 1 to 3) to make ECMP member decisions, one key is used for LAG to make LAG member decisions, and one key is used as a network key that can be stamped in network headers at the egress pipeline (for example, MPLS/PWE EL, VXLAN UDP port).

All load balancing configurations are optional, and default values are available to generate pseudo random keys after initialization.

13.2 Application Configuration Checklist

- Speculative parsing
- Symmetrical hashing configuration
- Layer records bits selection
- Seed selection
- Assign a CRC function to each load balancing client.
- Enable and disable header field usage in LB key generation

13.3 Speculative Parsing

Speculative parsing allows the device to speculate which network layer follows the MPLS BOS label if it is not terminated by the Tunnel-Termination stage. Speculative parsing also calculates the hash key for the speculated network layers. ‘

Speculative parsing is performed by classifying the first nibble (4 bits) after the MPLS BOS label. Most of the nibble values that follow the BOS MPLS labels are assumed to be either an Ethernet header first nibble or no header, where the selection between the two is determined by the default speculation configuration.

The nibble values in the following table can be configured either to the default speculation value (Ethernet header or no header) or to a specific protocol that is usually related to these nibbles.

Nibble Value	Configuration Option
0	Control word
1	Control word (GACh)
4	IPv4
5	MPLS BIER
6	IPv6

NOTE: Speculative parsing can be configured only while traffic is disabled.

It is possible to enable and disable speculative parsing per incoming port.

13.3.1 SOC Properties

None

13.3.2 Configuration Flow

The device is configured by default with nibbles 0x4 and 0x6 set to speculate IPv4 and IPv6 protocols, respectively, and all the other nibbles are set to *Ethernet*.

To change the default nibble speculation setting, call `bcm_switch_control_indexed_set(unit, key, value)`. The parameters are as follows:

- `key.type` – Set to `bcmSwitchMplsSpeculativeNibbleMap`.
- `key.index` – A supported nibble value.
- `value.value` – Supported `bcm_switch_mpls_next_protocol_t` option for the configured nibble.

The following table contains the supported nibble values and the supported configuration option for each nibble.

Nibble Value	Next Header Speculation Option
0,1	bcmSwitchMplsNextProtocolDefault – Use default configuration for this nibble
	bcmSwitchMplsNextProtocolControlWord – Speculate this nibble as control word
4	bcmSwitchMplsNextProtocolDefault – Use default configuration for this nibble
	bcmSwitchMplsNextProtocolIpv4 – Speculate as IPv4
5	bcmSwitchMplsNextProtocolDefault – Use default configuration for this nibble
	bcmSwitchMplsNextProtocolBierMpls – Speculate as MPLS BIER
6	bcmSwitchMplsNextProtocolDefault – Use default configuration for this nibble
	bcmSwitchMplsNextProtocolIpv6 – Speculate as IPv6
BCM_SWITCH_DEFAULT_NIBBLE_INDEX (16) Speculate option for all the nibbles except for nibbles 4 and 6. Nibbles 0, 1, and 5 use this speculation if their speculation is set to default.	bcmSwitchMplsNextProtocolEthernet – Speculate Ethernet by default
	bcmSwitchMplsNextProtocolNone – Do no speculate by default
BCM_SWITCH_DEFAULT_IPVX_NIBBLE_INDEX (17) Set the default speculation option for the IPvX nibbles (4 and 6).	bcmSwitchMplsNextProtocolEthernet – Speculate Ethernet by default
	bcmSwitchMplsNextProtocolNone – Do not speculate by default

Example: To associate the value 0x4 with IPv4 on unit, call:

```
bcm_switch_control_key_t key;
bcm_switch_control_info_t value;
key.type = bcmSwitchMplsSpeculativeNibbleMap;
key.index = 4;
value.value = bcmSwitchMplsNextProtocolIpv4;
bcm_switch_control_indexed_set(unit, key, value);
```

Speculative parsing can be enabled and disabled per incoming port.

Call `bcm_port_control_set()`, with type `bcmPortControlMplsSpeculativeParse`, to enable or disable the speculative parsing.

13.3.3 Shell Commands

None

13.3.4 Application Reference

None

13.4 Symmetrical Hashing Configuration

The following table describes the supported symmetric protocols.

Table 82: Symmetric Protocols

Protocol	Symmetric Fields
Ethernet	DA(6B)^SA (6B)
IPv4	DIP (4B)^SIP (4B)
IPv6	DIP (16B)^SIP (16B)
TCP/UDP/SCTP	SPORT (2B)^DPORT (2B)

The IPv4, ETH, and L4 protocols default symmetric options, as in Table 82, can be disabled using the `BCM_HASH_NON_SYMMETRICAL` flag in the header field enablers, see Section 13.8, Enable or Disable Header Field Usage in LB Key Generation.

Example:

If the Ethernet supports symmetric protocols, the two L2 packets (without any other header below the Ethernet header) with the MAC addresses shown in the following table will have the same generated LB key.

Packet Number	DA	SA
1	10:33:42:11:32:87	a9:b5:ff:cd:12:62
2	a9:b5:ff:cd:12:62	10:33:42:11:32:87

13.4.1 SOC Properties

TBD

13.4.2 Configuration Flow

None

13.4.3 Shell Commands

None

13.4.4 Application Reference

None

13.5 Bit Selection for Layer Records

Each layer record carries 32 bits of hashed header payload. Only 16 bits of that information (16 LSB or 16 MSB) can be used for the key generation.

Global configuration per load balancing client and layer record index is available (see Section 13.1, Introduction) to select which part of the hash key to use.

13.5.1 SOC Properties

None

13.5.2 Configuration Flow

All the layer records are set at initialization to use the 16 LSB bits.
To select the layer record bits that are taken for the key generation, call: `bcm_switch_control_indexed_set(unit, key, value);`

- `key.type` – Set to `bcmSwitchLayerRecordModeSelection`.
- `key.index` – The selected client induction that the layer records bit selection will be updated for using the following options:
  - `bcmSwitchECMPHashConfig`
  - `bcmSwitchECMPSecondHierHashConfig`
  - `bcmSwitchTrunkHashConfig`
  - `bcmSwitchECMPThirdHierHashConfig`
  - `bcmSwitchNwkHashConfig`
- `value.value` – A bitmap of 8 bits, one bit per layer recorded where 0 means selecting the 16 LSB bits and 1 for the 16 MSB bits.

13.5.3 Shell Commands

None

13.5.4 Application Reference

None

13.6 Seed Selection

Three main seed selection mechanisms can be globally configured:

- Each of the eight available CRC hashing functions uses a 16-bit seed number that affects the CRC function result.
- MPLS labels have another hash processing, in addition to the parser hash.
  The 32-bit seed of the MPLS hashing CRC function is configurable.
- The parser hashes each of the headers that are identified during the parsing stage into a layer record. The hash result per layer record is used to generate the load-balancing key. The seed value of this hash function is configurable.

13.6.1 SOC Properties

None

13.6.2 Configuration Flow

1. To set CRC function seeds, call: `bcm_switch_control_indexed_set(unit, key, value);`
   - `key.type` – Set to `bcmSwitchHashSeed`
   - `key.index` – The CRC function number, see Section 13.7, Assigning a CRC Function to the Load-Balancing Keys.
   - `value.value` – A 16 bit seed
2. To set the seeds for the MPLS stack hashing, call: `bcm_switch_control_set(unit, type, arg);`
   - `type` – `bcmSwitchMplsStack0HashSeed`
   - `arg` – A 32 bits seed.
3. To set the seeds for the parsing hashing stage, call: `bcm_switch_control_set(unit, type , arg);`
   - `type` – `bcmSwitchParserHashSeed`
   - `arg` – A 32 bits seed.

NOTE: It is recommended to leave the parser seed on the initialized value of 0. Other values might result in cases where a disabled layer has a different affect on the load balancing keys than if the layer was not on the received packet in the first place.

13.6.3 Shell Commands

None

13.6.4 Application Reference

None

13.7 Assigning a CRC Function to the Load-Balancing Keys

Each of the five load-balancing keys (ECMP hierarchies, LAG, and the network) should use a different CRC function.

13.7.1 SOC Properties

None

13.7.2 Configuration Flow

To set a CRC function to a client, call `bcm_switch_control_set(unit, type, arg);`

- `type` – Select one of the types in the following table.

Type	Description
`bcmSwitchECMPHashConfig`	ECMP-1
`bcmSwitchECMPSecondHierHashConfig`	ECMP-2
`bcmSwitchECMPThirdHierHashConfig`	ECMP-3
`bcmSwitchTrunkHashConfig`	LAG
`bcmSwitchNwkHashConfig`	Network

- `arg` – The CRC function number

13.7.3 Shell Commands

None

13.7.4 Application Reference

None

13.8 Enable or Disable Header Field Usage in LB Key Generation

The header field enablers provide the ability to exclude or include a certain header field from participating in the load-balancing key generation.

NOTE: The initial state of each field is enabled.

13.8.1 SOC Properties

None

13.8.2 Configuration Flow

To set which of the header fields should take part in the hashing, call `bcm_switch_control_set(unit, type, arg)`.

The `type` parameter is a supported header type, and `arg` is all the field flags that should remain for the load-balancing key generation.

The following table presents all the supported header types and each header supported fields.

Header Type	Supported Fields	Description
`bcmSwitchHashIP4OuterField`, `bcmSwitchHashIP4InnerField`	`BCM_HASH_FIELD_IPV4_ADDRESS`	IPv4 SIP and DIP
	`BCM_HASH_FIELD_PROTOCOL`	
	`BCM_HASH_NON_SYMMETRICAL`	Hash the IPv4 address in a non symmetrical way, must be set with the `BCM_HASH_FIELD_IPV4_ADDRESS` option
`bcmSwitchHashIP6Field`	`BCM_HASH_FIELD_IPV6_ADDRESS`	IPv6 SIP and DIP
	`BCM_HASH_FIELD_FLOW_LABEL`	
	`BCM_HASH_FIELD_NXT_HDR`	
`bcmSwitchHashL2OuterField`, `bcmSwitchHashL2InnerField`	`BCM_HASH_FIELD_MAC_ADDRESS`	SA and DA
	`BCM_HASH_FIELD_ETHER_TYPE`	The Ether type and all the VLANs TPIDs
	`BCM_HASH_FIELD_VLAN`	First and second VLAN ID
	`BCM_HASH_NON_SYMMETRICAL`	Hash the MAC address in a non-symmetrical way, must be set with the `BCM_HASH_FIELD_MAC_ADDRESS` option.
`bcmSwitchHashForceL2Field`	`True` or `False`	`True` indicates the terminated ETH fields will take part in hashing. `False` is the default value. It indicates the terminated ETH fields are not needed in hashing.
`bcmSwitchHashL4Field`	`BCM_HASH_FIELD_L4`	The TCP/UDP/SCTP source and destination ports
	`BCM_HASH_NON_SYMMETRICAL`	Hash the L4 ports in a non-symmetrical way, must be set with the `BCM_HASH_FIELD_L4` option
`bcmSwitchHashMPLSField0`	`BCM_HASH_MPLS_ALL_LABELS`	All the MPLS labels

NOTE: The GTP TEID field cannot be disabled from the load balancing key generation.

Example: To exclude the flow label from the IPv6 hashing fields, call:

```
bcm_switch_control_set(unit, bcmSwitchHashIP6Field, BCM_HASH_FIELD_IPV6_ADDRESS |
BCM_HASH_FIELD_NXT_HDR);
```

Example: To include terminated outer L2 SA and DA in hashing fields on unit0, call:

```
bcm_switch_control_set(0, bcmSwitchHashL2OuterField, BCM_HASH_FIELD_MAC_ADDRESS);
bcm_switch_control_set(0, bcmSwitchHashForceL2Field, True);
```

13.8.3 Shell Commands

None

13.8.4 Application Reference

None

13.9 Protocol Layer Disable from Hashing by LIF or Port

NOTE: This feature is not supported on the BCM88690, BCM88480 or BCM88800.

The following configuration allows omitting certain protocol layers from the load-balancing key-generation input for a given port or LIF.

Each set of disabled protocols is considered a unique profile. There are up to three profiles for the ports and up to three profiles for the LIFs besides the default profile where all the protocol layers are enabled.

If both the port and the LIF are using a non-default profile, the LIF profile is used.

13.9.1 SOC Properties

None

13.9.2 Configuration Flow

To remove protocols from participating in the hash for a given port use the following API:

```
bcm_switch_control_port_set(unit, port, type, flags);
```

- `port` – The incoming port that will use the configured protocol selection.
- `type` – `bcmSwitchHashLayersDisable`.
- `flags` – Any combination of the flags listed in Section 13.9.2.1, Flags

To remove protocols from participating in the hash for a given LIF use:

```
bcm_l3_ingress_create(unit, ing_intf, &l3if.l3a_intf_id)
```

- `l3if.hash_layers_disable` – Any combination of the flags listed in Section 13.9.2.1, Flags.

For more information about the `bcm_l3_ingress_create` API, see Chapter 22, IP Router.

13.9.2.1 Flags

The following flags remove protocols from participating in the load balancing key generation in the APIs described in the previous section.

- `BCM_HASH_LAYER_ETHERNET_DISABLE`
- `BCM_HASH_LAYER_IPV4_DISABLE`
- `BCM_HASH_LAYER_IPV6_DISABLE`
- `BCM_HASH_LAYER_MPLS_DISABLE`
- `BCM_HASH_LAYER_TCP_DISABLE`
- `BCM_HASH_LAYER_UDP_DISABLE`

13.9.3 Per-Box SEED

Networks typically configure a *SEED* per box to be used in the Load Balancing function to avoid *polarization*. For the polarization definition and examples see Section 13.9.3.1, Polarization Definition and Examples.

The SEED should be used as follows:

- The application is expected to perform some CRC function on the provided SEED, since SEEDs provided by network administrators tend to be *sequential*. Typically, the SEED is a 48b MAC Address or a 32b node ID.
- Use the CRC Result to determine *Bit Selection* [16LSBs, 16MSBs] per load_Balancing client [LAG,ECMP0…], per layer [0..7].
- Use the CRC Result to determine CRC function per load_Balancing client.

NOTE:

- Choose different bits of the CRC result for each configuration.
- Some packets have only a few layers, for example, PDUoIPv4oEth has only two layers. For these packets, the upper layers [16LSBs, 16MSBs] configuration does not affect the final LB_Key.
- Some packets use only a certain set of Load_Balancing clients, for example, a PDUoEth packet forwarding to a directly attached bridge over a LAG. For these packets, only the CRC configuration of the relevant Load_Balancing clients affect the LB_Key.

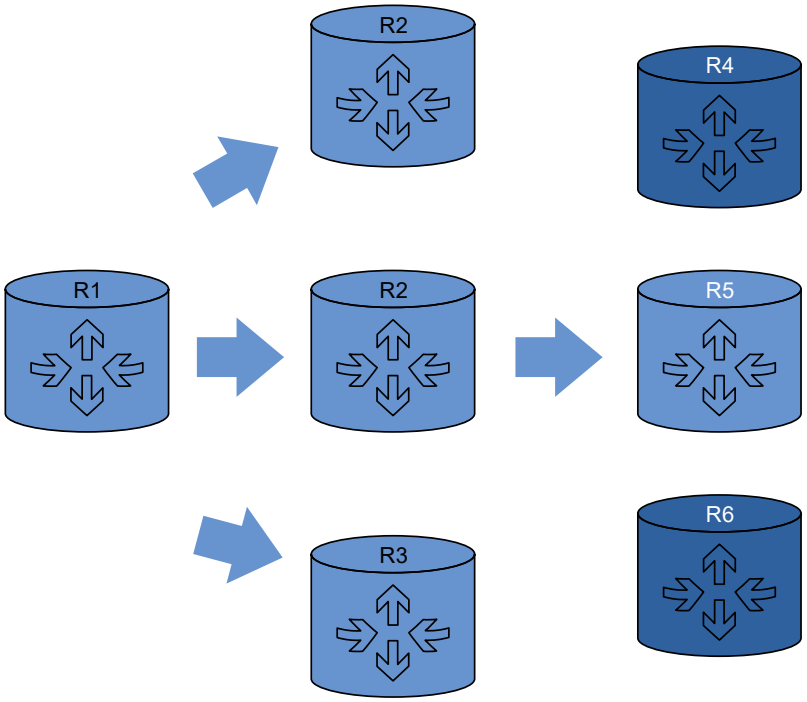
For more information on each of the configuration options, see the following sections:

- Section 13.5, Bit Selection for Layer Records
- Section 13.7, Assigning a CRC Function to the Load-Balancing Keys
- Section 13.4, Symmetrical Hashing Configuration

13.9.3.1 Polarization Definition and Examples

Using the same load balancing configuration for two connected devices can result in traffic polarization as seen in the following figure.

Figure 10: Traffic Distribution



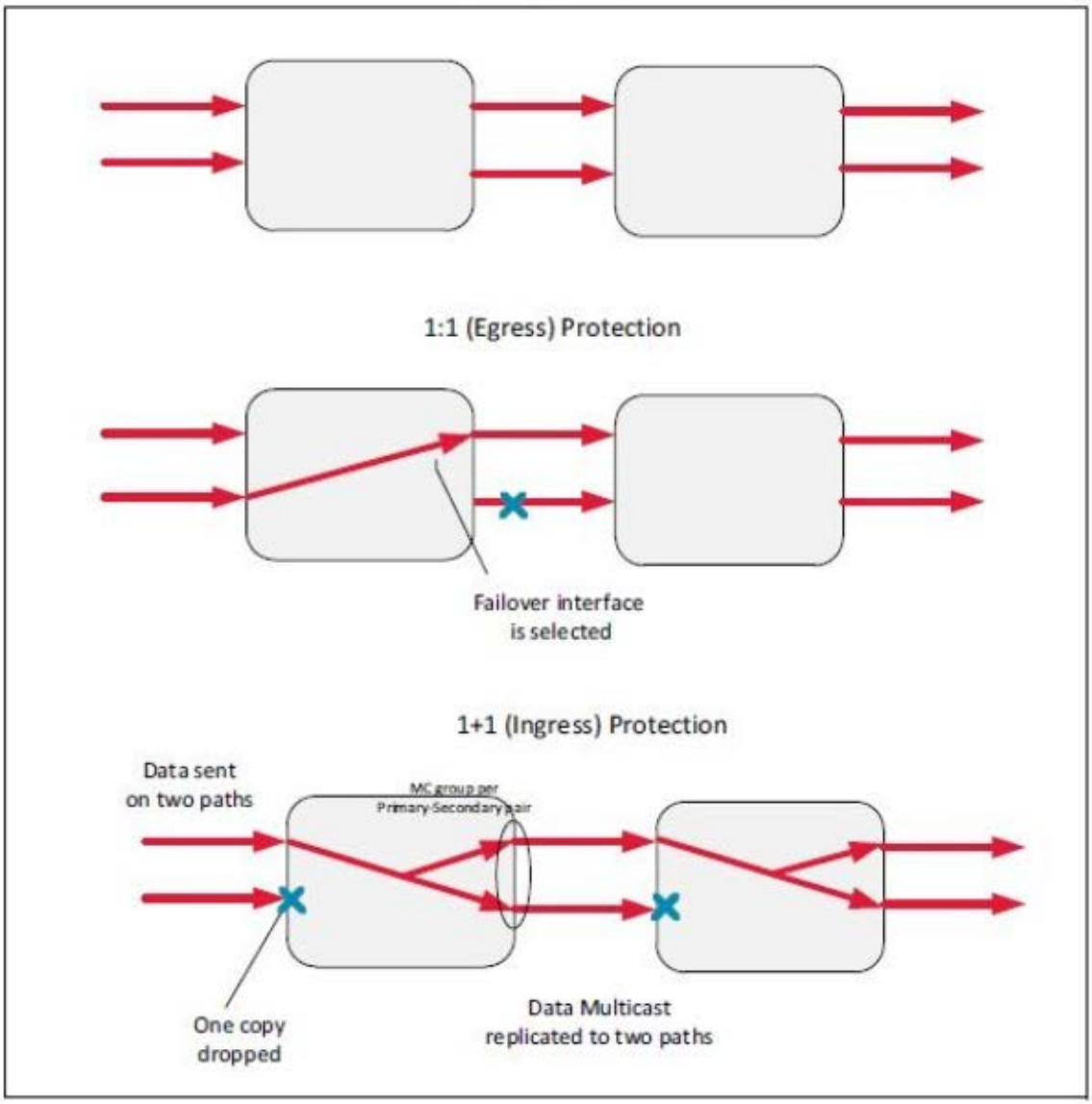
In this example, R1 and R2 share the same load-balancing configuration. The traffic that flows from R1 to R2 gets the same hashing keys on both devices, which leads to the same path selection and prevents R1 traffic that flows into R2 from reaching R4 and R6.

Chapter 14: Protection Switching

The device supports two types of protection schemes: egress 1:1 protection (also known as Head-End protection) and ingress 1+1 protection (also known as Tail-End protection). Protection schemes are a recurring theme in several L2 and L3 applications, for example MPLS, VPLS, and Queue-in-Queue. This introduction defines the relevant protection-switching terms and highlights the basic principles. The APIs that are called may be different from one application to another, but the calling sequence logic is the same.

Both types of protection schemes are described by the following figure.

Figure 11: Unicast Protection Schemes: Egress 1:1 and Ingress 1+1



14.1 Application Configuration Checklist

- VLAN-Port 1:1 Protection
  - Create a FEC Failover Object
  - Define a FEC object
  - Configure 1:1 UC protection on VLAN-Port LIFs
  - Create an Egress Failover Object
  - Configure 1:1 MC protection on VLAN-Port LIFs
- VLAN-Port 1+1 Protection
  - Create an Ingress Failover Object
  - Configure 1+1 protection on VLAN-Port LIFs
- MPLS-Port 1:1 Protection
  - Create a FEC Failover Object
  - Define a FEC object
  - Configure 1:1 UC protection on MPLS-Port LIFs
  - Create an Egress Failover Object
  - Configure 1:1 MC protection on MPLS-Port LIFs
- MPLS-Port 1+1 Protection
  - Create an Ingress Failover Object
  - Configure 1+1 protection on MPLS-Port LIFs
- MPLS-Tunnel 1:1 Protection
  - Create a FEC Failover Object
  - Define a FEC object
  - Configure 1:1 UC protection on MPLS-Tunnel LIFs
  - Create an Egress Failover Object
  - Configure 1:1 MC protection on MPLS-Tunnel LIFs
- MPLS-Tunnel 1+1 Protection
  - Create an Ingress Failover Object
  - Configure 1+1 protection on MPLS-Tunnel LIFs

14.2 Failover Object

The failover objects creation and IDs use the same BCM API, yet the failover objects and IDs are allocated in different spaces according to the selected protection scheme:

- Ingress – 1+1 protection
- FEC – Unicast 1:1 or N:1 protection
- Egress – Multicast 1:1 protection

It is possible to map multiple protected objects to the same failover object, allowing a single operation traffic switch-over for several objects that share the same forwarding protection scheme on the same device.

At failover object allocation, the failover status is initialized to pass traffic of pointing primary protected objects.

The SDK handles the FEC failover object allocation limitation. The failover object IDs allocation can also be managed manually. In such cases the relevant API verifies that the input aligns to what is stated above, including the mentioned limitations.

14.2.1 SOC Properties

None

14.2.2 Configuration Flow

1. Create Failover-ID Object by calling: `bcm_failover_create(unit, flags, failover_id)`
   – `flags` – The options are in the following table.

Flags	Description
`BCM_FAILOVER_WITH_ID`	Indicates `failover_id` as an input, specified and managed by the user. The following values are reserved and cannot be allocated by the user: ■ 0 – SDK *no-protection* indication ■ –1 – Hardware *no-protection* indication ■ BCM_FAILOVER_ID_LOCAL – Facility protection indication
`BCM_FAILOVER_FEC`	Allocate a failover object from the FEC failover space. If `BCM_FAILOVER_FEC_2ND_HIERARCHY` or `BCM_FAILOVER_FEC_3RD_HIERARCHY` are not set, the first failover-ID hierarchy level is assumed. The protection pointer hierarchy must be similar to the FEC hierarchy.
`BCM_FAILOVER_FEC_2ND_HIERARCHY`	Indicates the second hierarchy level (relevant only for a FEC failover object). Must be similar to the pointing FEC hierarchy.
`BCM_FAILOVER_FEC_3RD_HIERARCHY`	Indicates the third hierarchy level (relevant only for a FEC failover object). Must be similar to the pointing FEC hierarchy.
`BCM_FAILOVER_INGRESS`	Allocate a failover object from the ingress failover space.
`BCM_FAILOVER_ENCAP`	Allocate a failover object from the egress failover space.

2. To switch between Primary and Secondary traffic passing for objects that are protected with the failover object, call: `bcm_failover_set(unit, failover_id, enable)`
   – `failover_id` – The Failover object ID
   – `enable` – The selected interface to pass traffic:
   • 1 – Primary
   • 0 – Secondary
3. The failover object ID is made up of a failover type and a failover ID value. The following BCM API macros enable encoding and decoding of the failover object ID:
   – BCM_FAILOVER_SET – Make up a failover ID with failover type and failover ID value
   – BCM_FAILOVER_ID_GET – Retrieve the failover ID value
   – BCM_FAILOVER_TYPE_GET – Retrieve the failover type that has one of the following values:
   – BCM_FAILOVER_TYPE_INGRESS – Ingress 1+1 protection,
   – BCM_FAILOVER_TYPE_FEC – FEC Unicast 1:1/N:1 protection
   – BCM_FAILOVER_TYPE_ENCAP – Egress Multicast 1:1 protection

14.2.3 Shell Commands

None

14.2.4 Application Reference

See one of the protection schemes described in a later section.

14.3 FEC 1:1 Unicast Protection

FEC Protection implements 1:1 unicast protection for various L2/L3 applications.

The FEC Protection information is stored per a pair of FECs, referred to as a Super-FEC. A Super-FEC pair consists of an even FEC-ID and its successor. The FEC destination Primary/Secondary status is derived from their order with in the pair:

- Primary FEC – The even FEC-ID
- Secondary FEC – The odd FEC-ID

In case a protection pointer is defined for the Super-FEC pair, a FEC protection scheme is in place.

The SDK Super-FEC creation sequence assumes the Secondary FEC is created before the Primary-FEC.

There are three types of FEC protection:

- PATH – The failover status that is pointed by the protection pointer (primary/secondary) selects the active FEC within the pair.
- FACILITY – The active FEC within the super FEC is selected according to the state of the destination system port (up or down) rather than by the state of the failover object. Facility protection is enabled by setting a predefined protection pointer value that is smaller than the highest FEC-ID by one.
  Facility protection can be applied only by FEC entries that were allocated for the highest FEC hierarchy, the third hierarchy.

For FEC-ID allocation, see Chapter 9, FEC and ECMP Management.

NOTE: When a protected-FEC (part of a super-FEC) is created, it is not possible to modify the object to be unprotected, and vice versa. If this change is required, the user is expected to destroy both objects (primary and secondary) and re-create them as non-protecting.

In general, FEC Protection is complemented by Egress Protection as a 1:1 protection solution where MC Protection is achieved through Egress Protection.

When an L2/L3 object is created, it must indicate, in advance, if it is a 1:1 or 1+1 Unicast or Multicast protection object or unprotected (by failover-IDs fields non-zero indication). It is not possible to replace the object to be unprotected, and vice versa. If a change is required, it is expected to destroy both objects (primary and secondary) and re-create them.

14.3.1 Configuration Flow

1. Create a FEC Failover-ID by calling: `bcm_failover_create(unit, flags, &failover_id)`
   - `flags`: `BCM_FAILOVER_FEC` – Must be set.
   - `flags`: `BCM_FAILOVER_FEC_2ND_HIERARCHY`, `BCM_FAILOVER_FEC_3RD_HIERARCHY` – Depends on FEC hierarchy-level. For more information about the failover-ID object, see Section 14.2, Failover Object.

NOTE: Facility protection is only supported in the third hierarchy.

2. Create backup (secondary) protection FEC using the allocated protection pointer:
   `bcm_l3_egress_create(unit,flags,*egr,&if_id_secondary)`
   - `egr.failover_id` – The Failover-ID created on first step or by one of the following values:
     - 0 – No protection
     - BCM_FAILOVER_ID_LOCAL – Facility protection
   - `egr.failover_if_id` – Set to 0, Indicate object is secondary
   - `egr.destination` – For L2 protection: set the secondary L2 egress object (for example, VLAN-Port, MPLS-Port), if object-ID is already known. If not, update it later using the REPLACE flag.

- `egr.intf` – For L3 protection, set the secondary L3 egress object (for example, MPLS-Tunnel, IP-Tunnel) if object-ID is already known. If not, update it later using the REPLACE flag.
- For the other fields, see Section 22.8, L3 Egress Object: Ingress-FEC.
- FEC Object ID created, called `if_id_secondary` which is used in later steps.

3. Create primary protection FEC using the allocated protection pointer and the secondary FEC: `bcm_l3_egress_create(unit, flags,*egr,if_id_primary)`
   - `egr.failover_id` – The Failover-ID created on first step
   - `egr.failover_if_id` – `if_id_secondary`, indicate object is primary and bind it to the same Super-FEC.
   - `egr.intf` – For L2 protection, set the primary L2 egress object (for example, VLAN-Port, MPLS-Port) if the object-ID is already known. If not, update it later using the REPLACE flag.
   - `egr.intf` – For L3 protection, set the Primary L3 egress object (for example, MPLS-Tunnel, IP-Tunnel) if object-ID is already known. If not, update it later using the REPLACE flag.
   - For the other fields, see Section 22.8, L3 Egress Object: Ingress-FEC.
   - FEC Object ID created, called `if_id_primary` which is used in later steps.
4. Facility Protection: In Local/Facility or Path-and-Facility mode, to set active/inactive port call: `bcm_port_update(int unit, bcm_port_t port, int link)`;
   - `port` – The global system port, i.e., gport of `BCM_GPORT_TYPE_SYSTEM_PORT` type. Otherwise, the call has no effect.
   - `link` = 0 indicates port is Off. Otherwise On.

The main FEC sequence applies to L3 traffic and L2 traffic. For L2 traffic, it is also required to configure L2 objects (LIF objects) with learning information accordingly.

14.3.1.1 L2 Object: VLAN-Port 1:1 Unicast Protection

1. Create Secondary VLAN-Port by calling: `bcm_vlan_port_create(unit, vlan_port)`:
   - `vlan_port.failover_port_id` – Set with the Secondary FEC-ID (`if_id_secondary`). This will indicate VLAN-Port learning information is according to FEC-protection (Primary and Secondary).
   - For the other fields, see Section 27.4, Creating Service AC-LIFs.
   - VLAN-Port Object ID created, called `vlan_port_id_seconary` which can be used in `bcm_l3_egress_create` Secondary FEC as a destination.
2. Create Primary VLAN-Port by calling: `bcm_vlan_port_create(unit, vlan_port)`:
   - `vlan_port.failover_port_id` – Set with the Secondary FEC-ID (`if_id_secondary`). This will indicate VLAN-Port learning information is according to FEC- protection (Primary and Secondary).
   - For the other fields, see Section 27.4, Creating Service AC-LIFs.
   - VLAN-Port Object ID created, called `vlan_port_id_primary` which can be used in `bcm_l3_egress_create` Primary FEC as a destination.

14.3.1.2 L2 Object: MPLS-Port 1:1 Unicast Protection

The main FEC sequence applies to L3 traffic and L2 traffic. For L2 traffic, it is also required to configure L2 objects (LIF objects) with learning information accordingly:

1. Create secondary egress MPLS ports by calling `bcm_mpls_port_add(unit, vpn, &mpls_port)`;
   - `mpls_port_sec.flags` = `BCM_MPLS_PORT2_EGRESS_ONLY`;
   - For MPLS-Port fields, see Chapter 26, VPLS and VPWS.
   - MPLS-Port Egress Object ID created, called `mpls_port_sec.mpls_port_id` which can be used in `bcm_l3_egress_create` Secondary FEC as a destination.

2. Create primary egress MPLS ports by calling `bcm_mpls_port_add(unit, vpn, &mpls_port);`
   - `mpls_port_sec.flags` = `BCM_MPLS_PORT2_EGRESS_ONLY;`
   - For MPLS-Port fields, see Chapter 26, VPLS and VPWS.
   - MPLS-Port Egress Object ID created, called `mpls_port.mpls_port_id` which can be used in `bcm_l3_egress_create` Primary FEC as a destination.
3. Create secondary ingress MPLS ports by calling `bcm_mpls_port_add(unit, vpn, &mpls_port);`
   - `mpls_port.flags` = v;
   - `mpls_port.egress_tunnel_if` = FEC-Encoded interface. `BCM_L3_ITF_SET(encoded_itf, BCM_L3_ITF_TYPE_FEC, if_id_secondary);`
     It will set MPLS-Port learning information to match the Protection-FEC (primary and secondary).
   - MPLS-Port Ingress Object ID created, called `mpls_port_sec.mpls_port_id`. Usually it is easier to handle where ingress and egress have the same Global-LIF ID.
   - For MPLS-Port fields, see Chapter 26, VPLS and VPWS.
4. Create primary ingress MPLS ports by calling `bcm_mpls_port_add(unit, vpn, &mpls_port);`
   - `mpls_port.flags` = `BCM_MPLS_PORT2_INGRESS_ONLY;`
   - `mpls_port.egress_tunnel_if` = FEC-Encoded interface. `BCM_L3_ITF_SET(encoded_itf, BCM_L3_ITF_TYPE_FEC, if_id_secondary);`
     It will set MPLS-Port learning information to match the Protection-FEC (primary and secondary).
   - MPLS-Port Ingress Object ID created, called `mpls_port.mpls_port_id`. Usually it is easier to handle where ingress and egress have the same Global-LIF ID.
   - For MPLS-Port fields, see Chapter 26, VPLS and VPWS.

14.3.2 Shell Commands

None

14.3.3 Application Reference

Example of FEC protection

- **Type:** CINT reference
- **Path:** `$SDK/src/examples/sand/cint_mpls_3_level_protection.c`

14.4 VLAN-Port 1+1 Protection

When an L2/L3 object is created, it must indicate in advance if it is a 1:1 or 1+1 Unicast or Multicast protection object or not-protected (by the non zero indication in the failover-ID field). It is not possible to change the object to be non-protected, and vice versa. If this change is required, it is expected to destroy both objects (Primary and Secondary) and re-create them.

14.4.1 SOC Properties

None

14.4.2 Configuration Flow

1. Create an Ingress failover ID:

   `flags` = `BCM_FAILOVER_INGRESS`;.

   `bcm_failover_create(unit, flags, &failover_id)`

2. Create a secondary VLAN port:

   `vlan_port.ingress_failover_id – Ingress failover_id`;

   `vlan_port.ingress_failover_port_id = 0;` (indicate it is secondary)

   `vlan_port.failover_mc_group = failover_mc_group` (for learning information);

   `bcm_vlan_port_create(unit, &vlan_port);`

   It is not possible to update `failover_mc_group`. If required, the user must destroy the object and recreate it. Also, it is not possible to move from a protected to an unprotected object, and vice versa.

3. Create Primary VLAN port:

   `vlan_pri_port.ingress_failover_id = failover_id`;

   `vlan_pri_port.ingress_failover_port_id = 1`; (indicate it is primary)

   `vlan_pri_port.failover_mc_group = failover_mc_group`; (for learning information)

   `bcm_vlan_port_create(unit, vlan_pri_port);`

   It is not possible to update `failover_mc_group`. If required, the user must destroy the object and recreate it. Also, it is not possible to move from a protected to an unprotected object, and vice versa.

4. Create a multicast group consisting of the Primary and Secondary OutLIF:

   `bcm_multicast_create(unit, BCM_MULTICAST_GROUP_WITH_ID|BCM_MULTICAST_EGRESS_GROUP, &failover_mc_group)`

   `bcm_multicast_vlan_encap_get(unit,0,0, vlan_pri_port->vlan_port_id, &pri_encap_id)`

   `bcm_multicast_vlan_encap_get(unit,0,0, vlan_port->vlan_port_id, &encap_id)`

   `bcm_multicast_add(unit, BCM_MULTICAST_EGRESS_GROUP, failover_mc_group, 2 reps)`

   `rep` is an array of two replications with related `encap_id` and `port`;

14.4.3 Shell Commands

None

14.4.4 Application Reference

Example of VLAN-Port 1+1 protection

- **Type:** CINT reference
- **Path:** `$SDK/src/examples/sand/cint_ac_1to1_coupled_protection.c`

14.5 VLAN-Port 1:1 Multicast Protection

1:1 Multicast Protection is usually required in parallel to 1:1 Unicast Protection. For multicast traffic, the flow does not pass through the Protected-FEC, but rather through the Multicast-group. In that case, it is expected that the egress objects of primary and secondary will be added to the multicast replication group. One of the objects will pass traffic, the second will be dropped according to Failover-Status.

Following Configuration-flow, explain only the 1:1 Multicast Protection information required. For 1:1 Unicast protection information, see Section 14.3, FEC 1:1 Unicast Protection.

When an L2/L3 object is created, it is required to indicate in advance if it is a 1:1 or 1+1 Unicast or Multicast protection object or not-protected (by non-zero indications in failover-ID fields). It is not possible to change the object to be unprotected, and vice versa. If this change is required, it is expected to destroy both objects (Primary and Secondary) and re-create them.

14.5.1 SOC Properties

None

14.5.2 Configuration Flow

1. Create an Egress failover-ID:

   `flags` = `BCM_FAILOVER_ENCAP`;

   `bcm_failover_create(unit, flags, &failover_id)`

2. Create Secondary VLAN port:

   `vlan_port.egress_failover_id` – Egress Failover-ID created in step 1.

   `vlan_port.egress_failover_port_id = 0`; (indicate it is secondary)

   `vlan_port.flags` = `BCM_VLAN_PORT_EGRESS_PROTECTION`;

   `bcm_vlan_port_create (unit, &vlan_port);`

   The VLAN-Port is usually the same VLAN-Port created in the 1:1 Unicast.

   Protection flow (`vlan_port_id_secondary`). It is expected to add the information above as part of the object creation.

   It is not possible to move from a protected to an unprotected object, and vice versa.

3. Create the primary VLAN port:

   `vlan_pri_port.egress_failover_id` – The egress `failover_id` from step 1.

   `vlan_pri_port.egress_failover_port_id = 1`; (indicate it is primary)

   `vlan_port.flags` = `BCM_VLAN_PORT_EGRESS_PROTECTION`;

   `bcm_vlan_port_create (unit, 0, &vlan_pri_port);`

   The VLAN-Port is usually the same VLAN-Port created in the 1:1 Unicast.

   Protection flow (`vlan_port_id_secondary`). It is expected to add the information above as part of the object creation.

NOTE: It is not possible to move from protected to non-protected object and vice versa.

14.5.3 Shell Commands

None

14.5.4 Application Reference

Example of VLAN-Port protection

- **Type:** CINT reference
- **Path:** `$SDK/src/examples/sand/cint_ac_protection.c`

14.6 MPLS-Port 1+1 Protection

When an L2/L3 object is created, it is required to indicate in advance if it is 1:1/1+1 Unicast/Multicast protection object or not-protected (by failover-IDs fields non zero indication). It is not possible to replace the object to be non-protected, and vice versa. If this change is required, it is expected to destroy both objects (Primary and Secondary) and recreate them.

14.6.1 SOC Properties

None

14.6.2 Configuration Flow

1. Create an Ingress failover-ID:
   - `flags = BCM_FAILOVER_INGRESS;`
   - `bcm_failover_create(unit, flags, &failover_id)`
2. Create Secondary Ingress MPLS port:

   `mpls_port.ingress_failover_id` – Ingress `failover_id` from step 1.

   ```
   mpls_port.ingress_failover_port_id = 0;
   mpls_port.failover_mc_group = failover_mc_group;
   bcm_mpls_port_add(unit, vpn, &mpls_port);
   ```

   It is not possible to update failover_mc_group. If required, the user must destroy the object and recreate it. Also, it is not possible to move from a protected object to an unprotected object, and vice versa.
3. Create the Primary Ingress MPLS port:

   ```
   mpls_pri_port.ingress_failover_id = failover_id;
   mpls_pri_port.ingress_failover_port_id = 1;mpls_pri_port.failover_mc_group =
   failover_mc_group; bcm_mpls_port_add(unit, 0, &mpls_pri_port);
   ```

   It is not possible to update failover_mc_group. If required, the user must destroy the object and recreate it. Also, it is not possible to move from a protected object to an unprotected object, and vice versa.
4. Create egress MPLS ports, symmetrical to the Ingress MPLS ports, but unprotected. The egress MPLS ports will be used to get the encap ID for the multicast group in the next step.

5. Create a multicast group consisting of the Primary and Secondary OutLIF: `bcm_multicast_create(unit, BCM_MULTICAST_GROUP_WITH_ID|BCM_MULTICAST_EGRESS_GROUP, &failover_mc_group)`

   ```
   bcm_multicast_vlan_encap_get(unit,0,0, vlan_pri_port->vlan_port_id, &pri_encap_id)
   bcm_multicast_vlan_encap_get(unit,0,0, vlan_port->vlan_port_id, &encap_id)
   bcm_multicast_add(unit, BCM_MULTICAST_EGRESS_GROUP, failover_mc_group, 2 reps)
   ```

   `rep` is an array of two replications with related `encap_id` and `port`.

14.6.3 Shell Commands

None

14.6.4 Application Reference

TBD

14.7 MPLS-Port 1:1 Multicast Protection

When an L2/L3 object is created, it must indicate in advance if it is a 1:1 or 1+1 Unicast or Multicast protection object or not-protected (by nonzero indications in the failover-ID fields). It is not possible to change the object to be non-protected, and vice versa. If this change is required, it is expected to destroy both objects (Primary and Secondary) and re-create them.

14.7.1 SOC Properties

None

14.7.2 Configuration Flow

1. Create Egress failover-id:

   ```
   flags = BCM_FAILOVER_ENCAP;
   bcm_failover_create(unit, flags, &failover_id)
   ```

2. Create Secondary MPLS port:

   ```
   mpls_port.egress_failover_id = failover_id;
   mpls_port.egress_failover_port_id = 0;
   mpls_port.flags2 = BCM_MPLS_PORT2_EGRESS_PROTECTION;
   bcm_mpls_port_add(unit, 0, &mpls_port);
   ```

   It is not possible to move from a protected to an unprotected object, and vice versa. If required, destroy the object and recreate it.

3. Create the primary MPLS port:

   ```
   mpls_pri_port.egress_failover_id = failover_id;
   mpls_pri_port.egress_failover_port_id = 1;
   mpls_pri_port.flags2 = BCM_MPLS_PORT2_EGRESS_PROTECTION;
   bcm_mpls_port_add(unit, 0, &mpls_pri_port);
   ```

   It is not possible to move from a protected to an unprotected object, and vice versa. If required, destroy the object and recreate it.

14.7.3 Shell Commands

None

14.7.4 Application Reference

TBD

14.8 MPLS Tunnel 1+1 Protection

When an L2/L3 object is created, it must indicate in advance if it is a 1:1 or 1+1 Unicast or Multicast protection object or not-protected (by a nonzero indication in the failover-ID field). It is not possible to change the object to be non-protected, and vice versa. If this change is required, it is expected to destroy both objects (Primary and Secondary) and re-create them.

14.8.1 SOC Properties

None

14.8.2 Configuration Flow

Ingress Device

The 1+1 MPLS tunnel ingress device must be configured to filter one of two paths (primary/protected).

1. Create failover id group:

   ```
   flags = BCM_FAILOVER_INGRESS;
   bcm_failover_create(unit, flags, &failover_id)
   ```

2. Create a secondary MPLS tunnel:

   ```
   info.failover_id = failover_id;
   info.failover_tunnel_id = 0;
   bcm_mpls_tunnel_switch_create(unit, &vlan_port);
   ```

   It is not possible to move from a protected to an unprotected object, and vice versa. If required, destroy the object and recreate it.

3. Create the primary MPLS tunnel:

   ```
   pri_info.failover_id = failover_id;
   pri_info.failover_tunnel_id = 1; bcm_mpls_tunnel_switch_create(unit, &pri_info);
   ```

   It is not possible to move from a protected to an unprotected object, and vice versa. If required, destroy the object and recreate it.

Egress Device

1+1 egress MPLS tunnel device must be configured to send two copies (one for each path). To achieve that, create two OutLIF and add them to the dedicated multicast group. For more information, see Chapter 25, MPLS LER Encapsulation.

14.8.3 Shell Commands

None

14.8.4 Application Reference

TBD

14.9 MPLS Tunnel 1:1 Multicast Protection

When an L2/L3 object is created, it must indicate in advance if it is a 1:1 or 1+1 Unicast or Multicast protection object or not-protected (by nonzero indications in the failover-ID fields). It is not possible to replace the object to be non-protected, and vice versa. If the object changes, it is expected to destroy both objects (Primary and Secondary) and re-create them.

14.9.1 SOC Properties

None

14.9.2 Configuration Flow

1. Create failover id group:

```
flags = BCM_FAILOVER_ENCAP;
bcm_failover_create(unit, flags, &failover_id)
```

2. Create Protected MPLS tunnel port:

```
label_array.egress_failover_id = failover_id;
label_array.egress_failover_port_id = 0;
label_array.flags = BCM_MPLS_EGRESS_LABEL_PROTECTION;
bcm_mpls_tunnel_initiator_create(unit, intf, num_labels, *label_array);
```

   Note that all labels must have the same protection parameters. This example has only one label. For more information, see Chapter 25, MPLS LER Encapsulation.

3. Create the primary VLAN port:

```
label_array.egress_failover_id = failover_id;
label_array.egress_failover_port_id = 1;
label_array.flags = BCM_MPLS_EGRESS_LABEL_PROTECTION;
bcm_mpls_tunnel_initiator_create(unit, intf, num_labels, *label_array);
```

   Note that all labels must have the same protection parameters. This example has only one label. For more information, see Chapter 25, MPLS LER Encapsulation.

4. Select between the Primary/Backup LIF by calling:

```
bcm_failover_set(unit, failover_id, enable);
```

14.9.3 Shell Commands

None

14.9.4 Application Reference

TBD

14.10 API Descriptions

API Name	Highlights
`bcm_failover_create()`	Create FEC protection pointer.
`bcm_failover_destroy()`	Destroy FEC protection pointer.
`bcm_failover_set()`	Set protection pointer state.
`bcm_failover_get()`	Get protection pointer state.

For L2/L3 objects and Multicast APIs, see other sections.

Chapter 15: PP Statistics Generation

15.1 Introduction

PP Statistics is an important attribute in every network application. It enables counting and monitoring a variety of objects and behaviors. The statistics application is constructed of two main parts, packet processing that creates statistical information, and counter processor that maps the information into a counter ID. This section describes the handling of statistical information by the PP pipeline. In particular, it describes the linkage of a statistic object (PP object that requires statistic gathering) to a PP statistic information (stat-id, stat-pp-profile). It also contains information about metering properties of PP Objects.

NOTE: Trapped packets will not be counted and/or metered.

15.2 Application Configuration Checklist

- Configure counter processor database properties on device init. For details on counter and meter processing see the Counter Processor section in the *Traffic Manager Programming Guide*.
- Optional: For egress metering: add QoS meter offset.
- Configure statistics pp profiles
- Create Object for counting
- Configure Stat-Metadata

15.3 Egress QoS Meter Offset

For egress meter, it is possible to add a QoS meter offset to the base meter pointer according to Traffic-class. In that case, Meter-point will be the result of object-stat-id and QoS-meter-Offset. QoS meter offset is an offset per TC.

This functionality is decided according to stat-pp-profile.

15.3.1 SOC Properties

None

15.3.2 Configuration Flow

Configuration of this offset is made by QoS APIs. By default, offset is zero for all TCs.

1. Create QoS profile by calling: `bcm_qos_map_create(unit, flags, &map_id);`
   - `flags` = `BCM_QOS_MAP_EGRESS | BCM_QOS_MAP_POLICER`.
   - `map_id` – ID allocated by the driver
2. Per traffic class, add new entry associated to the map id using `bcm_qos_map_add(unit, flags, &map, map_id);`
   - `flags = BCM_QOS_MAP_EGRESS | BCM_QOS_MAP_POLICER`
   - `map_id` – map_id received from `bcm_qos_map_create`
   - `map.int_pri` – TC to map [0..7]
   - `map.policer_offset` – Required offset [0..7]
3. Associate Meter QoS profile to stat-pp profile, see Chapter 15.4, Statistics PP Profile.

15.3.3 Shell Commands

None

15.3.4 Application Reference

Example of Policer offset:

- **Type:** CINT reference
- **Path:** `$SDK/src/examples/dnx/cint_policer_generic_meter.c`
- **Function**: `policer_egress_pointer_generation`

15.4 Statistics PP Profile

PP Statistics profile is used to set mutual properties of Statistic commands. There are statistic profiles for IRPP and ETPP stages. The profile is a 5 bit property of the statistic object. For each stage, these properties might be different as described below. .

In IRPP:

- Statistics-Interface
- Statistics-Object-Type
- Counter enabled indication
- Meter enabled indication
- Statistic command is part of PMF second-stage program selection indication
- Ingress metadata properties (size + start bit)

In ETPP:

- Statistics-Interface
- Statistics-Object-Type
- Meter enabled indication, if yes:
  - Meter-Interface
  - Meter Qos-Profile

15.4.1 SOC Properties

None

15.4.2 Configuration Flow

To allocate and configure statistics pp profile, call:

```
bcm_stat_pp_profile_create(unit, flags, engine_source, stat_pp_profile,
stat_pp_profile_info);
```

- `engine_source` – Counting source, one of `IngressReceivePp, EgressTransmitPp`.
- `stat_pp_profile` – PP Stat profile allocated or provided by the user
- `flags`:

Supported flags	Description
`BCM_STAT_PP_PROFILE_WITH_ID`	Use ID provided by user
`BCM_STAT_PP_PROFILE_REPLACE`	Override existing configuration

`stat_pp_profile_info`:

- `stat_pp_profile_info.counter_command_id` – Statistics-Interface as defined in `bcm_stat_counter_interface_t`

  In ingress, it can be both a counter and a meter. The interface mapping is shown in the following table:

Interface	Internal Counter	Metering	Loss Measurement
Ingress interface	{0..9}	{0..2}	{7..9}
Egress interface	{0..5}	{0..1}	{0..2}

- `stat_pp_profile_info.stat_object_type` – Statistics-Object-Type in the range of 0 to `BCM_STAT_MAX_NUMBER_OF_OBJECT_TYPES-1`
- `stat_pp_profile_info.is_meter_enable` – For meter should be 1, for counter should be 0.
- `stat_pp_profile_info.meter_command_id` – Meter command, if meter is enabled.
- `stat_pp_profile_info.meter_qos_map_id` – Meter QoS map id received by `bcm_qos_map_create` (see Section 15.3, Egress QoS Meter Offset).
- `stat_pp_profile_info.is_fp_cs_var` – Enable PMF2 context selection
- `stat_pp_profile_info.ingress_tunnel_metadata_size` – Ingress tunnel stage metadata size in the global metadata buffer.
- `stat_pp_profile_info.ingress_forward_metadata_size` – Forwarding stage metadata size in the global metadata buffer.
- `stat_pp_profile_info.ingress_tunnel_metadata_start_bit` – Ingress tunnel stage metadata start bit the in global metadata buffer.
- `stat_pp_profile_info.ingress_forward_metadata_start_bit` – Forwarding stage metadata size in the global metadata buffer.
- `stat_pp_profile_info.ingress_fwd_plus_one_metadata_size` – The same as `ingress_fwd_metadata_size`, for fwd plus one.
- `stat_pp_profile_info.ingress_fwd_plus_one_metadata_start_bit` – The same as `ingress_fwd_metadata_start_bit`, for fwd plus one.

15.4.3 Application Reference

Example of `stat_pp` user applications:

- **Type:** CINT reference
- **Path:** `$SDK/src/examples/dnx/stat_pp`

15.4.4 Statistics Object

Statistics command can be generated for most objects created by the user. There are three types of PP objects that may get a statistic command: Forwarding, LIF, and Port. To enable statistic command generation on a PP object, it should be associated with a statistic id and a valid statistics PP profile. The statistics ID should be pre-allocated, as described in the Counter Processor section in the BCM88690 *Traffic Manager Programming Guide*. The statistics PP profile should be preallocated as described previously in this section.

15.4.5 Statistics Metadata

Metadata consists of additional packet attributes for statistics set calculations (such as traffic class). Without metadata, the object can be counted only according to the information on the lookup result itself (stat_id and command). The metadata is additional data that is not part of the result (coming from the packet header), which allows distinguishing attributes inside the counter set. Different metadata will result in different counter offsets.

15.4.6 SOC Properties

None

15.4.7 Configuration Flow

Forwarding Objects

A statistic command may be created to the following forwarding objects:

Forwarding Object	API to Create the Forwarding Object
MAC table entry	`bcm_l2_addr_add`
MPLS LSR entry	`bcm_mpls_tunnel_switch_create`
IP host entry	`bcm_l3_host_add`
IP MC entry	`bcm_ipmc_add` Supported only when used in full mask when using internal memory (LEM)
FEC	`bcm_l3_egress_create`
ECMP	`bcm_l3_egress_ecmp_create`
VSI	`bcm_vlan_control_vlan_set`

To set a statistic command on a forwarding object, user should use the following fields present in each of these APIs:

- `stat_id` – Statistics object id, should be in valid range configured by CRPS APIs. For more information, see the Counter Processor section in the BCM88690 *Traffic Manager Programming Guide*.
- `stat_pp_profile` – Statistics PP profile

For `bcm_vlan_control_vlan_set`, the API is used for both ingress and egress configuration, thus the following fields should be used to distinguish between them:

- `ingress_stat_id` – Ingress statistics object id
- `ingress_stat_pp_profile` – Ingress statistics pp profile
- `egress_stat_id` – Egress statistics object id
- `egress_stat_pp_profile` – Egress statistics pp profile

For `bcm_l3_egress_create()` with the `BCM_L3_INGRESS_ONLY` flag for creating FECs with protection (see Chapter 14, Protection Switching), the following guidelines apply:

- The backup FEC stat ID must be configured to 0.
- The primary FEC stat ID should be configured to the desired stat ID.
- The primary FEC is counted on {configured stat ID – 1}.
- The backup FEC is counted on {configured stat ID – 2}.

LIF Objects

LIF objects are created in termination and encapsulation stages. A statistic command may be created to the following LIF objects:

LIF Object	API to Create the LIF Object	Enable Flag
VLAN PORT	`bcm_vlan_port`	`BCM_VLAN_PORT_STAT_INGRESS_ENABLE` – Ingress `BCM_VLAN_PORT_STAT_EGRESS_ENABLE` – Egress
MPLS PORT	`bcm_mpls_port`	`BCM_MPLS_PORT2_STAT_ENABLE`
MPLS LER INGRESS TUNNEL	`bcm_mpls_tunnel_switch`	`BCM_MPLS_SWITCH2_STAT_ENABLE`
IP INGRESS TUNNEL	`bcm_tunnel_terminator`	`BCM_TUNNEL_TERM_STAT_ENABLE`
MPLS LER EGRESS TUNNEL	`bcm_mpls_tunnel_initiator`	`BCM_MPLS_EGRESS_LABEL_STAT_ENABLE`
IP EGRESS TUNNEL	`bcm_tunnel_initiator`	`BCM_TUNNEL_INIT_STAT_ENABLE`
OUT RIF	`bcm_l3_intf_create`	`BCM_L3_FLAGS2_EGRESS_STAT_ENABLE`
ARP	`bcm_l3_egress_create`	`BCM_L3_FLAGS2_EGRESS_STAT_ENABLE`
IN RIF	`bcm_vlan_control_vlan_set`	—

NOTE:

- To enable statistics command generation on a LIF object, set the enable flag upon creation.
- The IN-RIF object uses VSI statistics parameters that were created by `bcm_vlan_control_vlan_set`.

To set statistics command generation on a LIF object, user should use the following API:

- `bcm_gport_stat_set(unit, gport, core_id, engine_source, stat_info);`
  - `gport` – gport representing the LIF object, output of the LIF creation APIs above
  - `engine_source` – Counting source stage. May get one of the following enums:
    - IRPP – `IngressReceivePp`
    - ETPP – `EgressTransmitPp`.
  - `stat_info.stat_id` – Statistics object id, should be in valid range configured by CRPS APIs. For more information, refer to the *Traffic Manager Programming Guide*.

NOTE: The use of `bcm_gport_stat_set` for LIF objects is enabled only for objects that were created or replaced with their matching stat enable flag. The only exceptions are SRv6 OutLIF tunnels and virtual VLAN-Port objects. For these cases, the operation may succeed (even when no flag was raised) if the selected LIF format contains the stat information fields. For SRv6 and virtual objects, this behavior may change in future releases to validate the stat enable flag for these cases as well.

  - `stat_info.stat_pp_profile` – Statistics PP profile.
- For the Out-RIF count (above), the `bcm_l3_intf_create` API members `stat_id` and `stat_pp_profile` are used. There is no need to call `bcm_gport_stat_set()`.

To get statistics command generation on a LIF object, use the following API:

```
bcm_gport_stat_get(unit, gport, core_id, engine_source, *stat_info);
```

The parameters are as follows:

- `gport` – A gport representing the LIF object, input from the LIF creation APIs described previously.
- `engine_source` – Counting source stage, which can be one of the following enums:
  - IRPP – IngressReceivePp
  - ETPP – EgressTransmitPp.
- `stat_info` – Statistics PP information structure to be filled by the API (`stat_id` and `stat_pp_profile`)

NOTE: BCM88690 only: Egress VSI statistics are not supported in BCM88690.

- Metadata

  Ingress metadata

  The following table summarizes metadata countable objects and metadata types.

Object	Statistics Metadata
InLIF	Next layer type, cast
Ingress forward domain (VSI, VRF)	■ Current layer type, cast, known or unknown ■ Indication of next layer type, cast

  - Metadata is written to a global buffer that is passed along the pipe for processing.
  - The buffer field size and offset are managed by the user.
  - When creating a stat PP profile using `bcm_stat_pp_profile_create()` to configure the metadata size and shift per profile in `bcm_stat_pp_profile_info_t`, configure the following parameters:
    - `ingress_tunnel_metadata_size`
    - `ingress_tunnel_metadata_start_bit`
    - `ingress_forward_metadata_size`
    - `ingress_forward_metadata_start_bit`
    - `ingress_fwd_plus_one_metadata_size`
    - `ingress_fwd_plus_one_metadata_start_bit`
  - The main usage of the InLIF metadata is for XXXoETH. The expansion works only for the AC statistics.
  - This configuration is per profile and can be overrun per packet if not configured correctly (that is, a packet hits two statistic objects with metadata configured, and the attributes of both profiles point to the same location inside the buffer).

Metadata API

- `bcm_stat_pp_metadata_set(unit, bcm_stat_pp_metadata_info);`

  `bcm_stat_pp_metadata_info` – A struct for picking the relevant metadata attributes, which include the following:

  - `flags` – Including ingress and egress, metadata type (that is, fwd, fwd+), and so on. Refer to `BCM_STAT_PP_METADATA_XXX`.

    `BCM_STAT_PP_METADATA_TUNNEL` – Set metadata according to the terminated objects at ingress tunnel termination, or according to egress tunnel encapsulation objects

    `BCM_STAT_PP_METADATA_FORWARD` – Set metadata according to forwarding layer type

    `BCM_STAT_PP_METADATA_FORWARD_PLUS_ONE` – Set metadata according to the next layer after forwarding.
  - `header_type` – Header type of the packet. This can be the current layer type or the next layer type, depending on the object to be counted (see the preceding table). See `bcm_stat_pp_metadata_header_type_t`.
  - `address_type_flags` – Refer to `BCM_STAT_PP_METADATA_ADDRESS_TYPE_XXX`. For ingress metadata, if `BCM_STAT_PP_METADATA_TUNNEL` or `BCM_STAT_PP_METADATA_FORWARD_PLUS_ONE` is set, only the `BCM_STAT_PP_METADATA_ADDRESS_TYPE_MCAST` flag is supported.
  - `metadata` – The metadata value to use (indicates the counter offset inside the `counter_set`).

  Upon hit in an object with statistics and metadata configured on it, the updated counter is the base counter (CRPS) + metadata value offset (as was set be the user in the last above field).

Egress metadata

Metadata countable objects and metadata type:

Object	Statistics Metadata
Egress forward domain (VSI, VRF)	Header type, cast
OutLIF	Header type, cast (cast only supports IPvX UC/MC)

- Metadata is written to a global buffer that is passed along the pipe for processing
- A dedicated API (`bcm_stat_counter_expansion_select_set()`) handles the offset and size inside the buffer. See Section 33.20.2, Counter Resource Management.

15.4.8 Application Reference

Example of `stat_pp` user applications:

- **Type:** CINT reference
- **Path:** `$SDK/src/examples/dnx/stat_pp`

15.4.9 API Descriptions

API name	Highlights
`bcm_stat_pp_profile_create()`	Maps statistic profile to profile properties
`bcm_stat_pp_profile_get()`	Get statistic profile information according to profile id
`bcm_stat_pp_profile_delete()`	Remove a statistic profile according to the profile id
`bcm_gport_stat_set()`	Maps physical or logical ports to stat id and stat profile
`bcm_gport_stat_get()`	Get a mapping of physical or logical port to stat id and stat pp profile
`bcm_stat_pp_metadata_set()`	Configures metadata attributes
`bcm_stat_pp_metadata_get()`	Get metadata value according to defined metadata header type

Chapter 16: VLAN Editing Mechanism

16.1 Introduction

VLAN editing enables manipulation of the Ethernet VLAN tags for any applicable application. An editing action is selected for the ingress forwarding Ethernet header and again at the egress for the same header and for the Ethernet Link Layer if encapsulation is applied. Whenever the edited Ethernet header is not the Link Layer Ethernet, it will be referred as a Native Ethernet VLAN Editing, both at ingress and at egress.

The VLAN Editing action is selected according the incoming Ethernet header at the appropriate stage and according to the underlying layer—the incoming/outgoing VLAN-Domain for Link Layer VLAN editing, and according to the underlying incoming/outgoing tunnel for native VLAN editing.

The relevant Ethernet header is assessed according to up to two of its outermost VLAN tags. The TPIDs in those VLAN tags are parsed and mapped into an incoming tag structure. The VLAN values of the VLAN tags affect the resolved InLIF or OutLIF. Those LIFs store a VLAN Edit Profile value that can possibly be seen as representing the structure of the outgoing VLAN tags.

The resolved action at each VLAN editing stage is the result of a mapping between the incoming tag structure and the VLA Edit Profile.

16.2 Application Configuration Checklist

The VLAN editing configuration checklist includes the following:

- TPID Definitions, as described in Section 21.3, TPID Definitions.
- Tag format classification
- VLAN Edit attributes per AC-LIF
- VLAN Edit command mapping
- VLAN Edit command setting

16.3 Tag Format Classification

A tag format is the classification result of the two outermost tag TPIDs of the handled Ethernet header. Classification of a combination of the two outermost tag TPIDs constitutes one or more entries that are part of the whole Classification mapping per the underlying object (port for Link Layer Ethernet headers or Tunnel InLIF VLAN-Domain for Native Ethernet headers). The amount of Classification mappings in a device is limited and managed by the SDK.

The tag format serves two main functions:

- Mapped together with a VLAN-Edit Profile to select a VLAN-Editing command-id.
- At Ingress – Selects the performed InLIF lookup type.

A tag format requires creation prior to being used for Ethernet packet classification. The creation phase provides the user with a Tag format that is associated to a specific LIF lookup type that is supplied by the user. A created Tag format can later be used by multiple Ethernet classifications regardless of the underlying object. The SDK maintains a predefined amount of Tag formats per LIF lookup type, as described later in this section. All unused Tag formats should be cleared.

A classification that results in a created Tag format, is identified by a preceding parser output that consists of up to two defined global TPIDs, as well as a Priority-Tag indication that is set whenever a tags VLAN-ID is zero. Any encountered TPID that is not one of the configured global TPIDs is treated for classification as if the tag is empty.

Per classification, additional attributes can be configured besides a Tag Format:

- Acceptable frame type – Drop a packet with specific TPIDs in the Ethernet header.
- Determine which of the tags is a C-Tag/S-Tag.

The configured Tag format is an SDK handled value that is mapped internally to one of the HW tag-structure values.

Tag format 0 is a reserved value for Untagged packets and is the only value that is allowed for Untagged packets.

The Tag format classification is dependent on the global TPIDs configuration. Any change in the global TPIDs may require reconfiguration of the Tag format classification. In the same way, creation or deletion of Tag formats may require reconfiguration of VLAN Edit command mappings.

During SDK initialization, a few reset classification mappings are allocated with specific values that can be seen in “DNX Device data”:

- Default native for IVE
- Default native for EVE

All VLAN-Domains (for ingress native ACs) are associated with the Default IVE. Tag format ‘0’ is created during initialization for the benefit of untagged packets. For suggested additional default configurations, see Section 16.3.4, Application Reference.

The above default classifications can be modified; or another classification mapping for each underlying object can be used.

In the BCM8869X generation of devices, as opposed to previous generations, Tag-structure is not only used for resolving IVE commands but also used for ingress match lookup type selection (that is, to decide which VLAN-port lookup to perform). For more information, refer to the LLVP Initialization and Tag Format Management section in the *Packet Processing Software Backward Compatibility with the BCM88670 and BCM88680* application note (88670-88690-AN2xx). This change requires the user to align between the packet-type and the range of values that can be used. Those ranges also apply to egress (while `tag_structure` is not used for the lookup match type).

The mechanism for allocating the SW-ID tag structure and mapping it to the desired HW tag-structure value, based on the tag-structure type to set (untagged, s_tag, c_tag, and so on) is as follows:

- `bcm_port_tpid_class_create` – Create a SW-ID value for a tag-structure and map it to the user-requested tag_structure HW value. It can be allocated for ingress-only, egress-only, or symmetric. The SW-ID value can be defined by the user (WITH_ID flag) or dynamically allocated. Each tag-structure has a different amount of unique IDs that can be allocated but is typically two or four IDs per tag-structure.
- `bcm_port_tpid_class_destroy` – Free the SW-ID value of for a tag-structure that was created previously by `bcm_port_tpid_class_create`. Removes the mapping to the `tag_structure` HW value.

16.3.1 SOC Properties

None

16.3.2 Configuration Flow

The configuration flow is as follows:

1. Create a tag format (if not created) by calling the creation API:
   `bcm_port_tpid_class_create(unit, flags, tag_structure_type, tag_format_class_id)`
   - `flags`
     - `BCM_PORT_TPID_CLASS_INGRESS_ONLY` – The allocated Tag format is applicable only for the Ingress side.
     - `BCM_PORT_TPID_CLASS_EGRESS_ONLY` – The allocated Tag format is applicable only for the Egress side.
     - `BCM_PORT_TPID_CLASS_WITH_ID` – The user specifies an unallocated Tag format in the field `tag_format_class_id`.
   - `tag_structure_type`
     - `bcmTagStructTypeUntag`
     - `bcmTagStructTypeSTag`
     - `bcmTagStructTypeCTag`
     - `bcmTagStructTypeSCTag`
     - `bcmTagStructTypeSPrioCTag`
     - `bcmTagStructTypeSSTag`
     - `bcmTagStructTypeCCTag`
     - `bcmTagStructTypeSPrioTag`

     For more information see, Section 21.6, Port x VLAN-Tag-Structure Configuration within Chapter 21, Ethernet Bridge
   - `tag_format_class_id` – The retrieved tag format. For `BCM_PORT_TPID_CLASS_WITH_ID` usage, this should also be a user supplied value in the accepted range.

     To free an allocated Tag format use `bcm_port_tpid_class_destroy()`.

2. To configure a classification per port, call the classification setting API:
   `bcm_port_tpid_class_set(unit, tpid_class)`
   - tpid_class.flags
     - `BCM_PORT_TPID_CLASS_INGRESS_ONLY` – The Tag format setting is applicable only for the ingress side. If no ingress native flag is supplied, it refers to configuration per the Ethernet header's underlying object.
     - `BCM_PORT_TPID_CLASS_EGRESS_ONLY` – The Tag format setting is applicable only for the egress side. If no egress native flag is supplied, it refers to the configuration per the Ethernet header's underlying object.
     - `BCM_PORT_TPID_CLASS_NATIVE_IVE` – Tag format setting for the Native IVE default. Can only be configured by itself or symmetrically with the Native EVE.
     - `BCM_PORT_TPID_CLASS_NATIVE_EVE` – Tag format setting for the Native EVE default to be used by all tunnels. Can only be configured by itself or symmetrically with the Native IVE.
     - `BCM_PORT_TPID_CLASS_DISCARD` – Discard the packet for the specified TPIDs combination. It can be applied for ingress or egress by using `BCM_PORT_TPID_CLASS_INGRESS_ONLY/EGRESS_ONLY` together.
     - `BCM_PORT_TPID_CLASS_OUTER_IS_PRIO` – Limits the setting to outer priority tag classification entries.
     - `BCM_PORT_TPID_CLASS_OUTER_NOT_PRIO` – Limits the setting to outer non-priority tag classification entries.
     - `BCM_PORT_TPID_CLASS_OUTER_C` – Determine the Outer tag is a C-Tag. If an Outer Tag exists and this flag is off, the tag is treated as an S-Tag.
     - `BCM_PORT_TPID_CLASS_INNER_C` – Determine the Inner tag is a C-Tag.
   - `tpid_class.port` – A gport the represents the underlying object:
     - Physical Port – Link Layer Ethernet header.
     - Tunnel – Native Ethernet header over an IP or MPLS Tunnel.
     - MPLS Port – Native Ethernet header over PWE.

  - 0 – When setting Native default and Egress Tunnel classifications (`BCM_PORT_TPID_CLASS_NATIVE_IVE`, `BCM_PORT_TPID_CLASS_NATIVE_EVE`).
- `tpid_class.tpid1` – The TPID of the outer tag in the handled Ethernet header. One of the configured global TPIDs.
  - `BCM_PORT_TPID_CLASS_TPID_INVALID` – Represents an Outer tag TPID that is not one of the global TPIDs or no existing Outer Tag.
  - `BCM_PORT_TPID_CLASS_TPID_ANY` – Represents a similar configuration for all Outer tag TPIDs. This option sets multiple entries in the same classification mapping. For example, when used with a valid `tpid_class.tpid2`, any packet with an inner tag TPID that equals `tpid_class.tpid2` will be classified to the specified Tag format.
- `tpid_class.tpid2` – The TPID of the inner tag in the handled Ethernet header.
  - One of the configured global TPIDs
  - `BCM_PORT_TPID_CLASS_TPID_INVALID` – Represents an Inner tag TPID that is not one of the global TPIDs or no existing Inner Tag.
  - `BCM_PORT_TPID_CLASS_TPID_ANY` – Represents a similar configuration for all Inner tag TPIDs. This option sets multiple entries in the same classification mapping.
- `tpid_class.tag_format_class_id` – A classifier Tag format that was previously allocated.

16.3.3 Shell Commands

None

16.3.4 Application Reference

Refer to `src/appl/reference/dnx/appl_ref_vlan_init.c`. In particular, see the TPID initialization function `appl_dnx_global_tpid_init()`.

NOTE: Global TPIDs must be set prior to setting LLVP profile.

See LLVP initialization function `appl_dnx_vlan_llvp_init()`:

- At application initialization, after global TPID values are set, both default LLVP profiles are set for the following packets:
  - Untag.
  - One tag – C tag.
  - One tag – S tag.
  - Double tag – S tag plus C tag.
  - Double tag with priority – S tag with priority (VID =0) plus C tag.
  - Double tag – S tag plus S tag.
  - Double tag – C tag plus C tag.

  The following table summarizes the tagging.

[IN] Outer TPID Value	[IN] Inner TPID Value	[IN] Outer Is Priority	[OUT] LLVP Incoming Tag-Structure
`BCM_PORT_TPID_CLASS_TPID_INVALID`	`BCM_PORT_TPID_CLASS_TPID_INVALID`	No	`DNX_PP_ETHERNET_FRAME_VLAN_FORMAT_NONE`
`DNX_TPID_VALUE_C_8100_TAG`	`BCM_PORT_TPID_CLASS_TPID_INVALID`	No	`DNX_PP_ETHERNET_FRAME_VLAN_FORMAT_C_TAG`
`DNX_TPID_VALUE_S_9100_TAG`	`BCM_PORT_TPID_CLASS_TPID_INVALID`	No	`DNX_PP_ETHERNET_FRAME_VLAN_FORMAT_S_TAG`

[IN] Outer TPID Value	[IN] Inner TPID Value	[IN] Outer Is Priority	[OUT] LLVP Incoming Tag-Structure
DNX_TPID_VALUE_S_88A8_TAG	BCM_PORT_TPID_CLASS_TPID_INVALID	No	DNX_PP_ETHERNET_FRAME_VLAN_FORMAT_S_TAG
DNX_TPID_VALUE_S_9100_TAG	DNX_TPID_VALUE_C_8100_TAG	No	DNX_PP_ETHERNET_FRAME_VLAN_FORMAT_S_C_TAG
DNX_TPID_VALUE_S_88A8_TAG	DNX_TPID_VALUE_C_8100_TAG	No	DNX_PP_ETHERNET_FRAME_VLAN_FORMAT_S_C_TAG
DNX_TPID_VALUE_S_9100_TAG	DNX_TPID_VALUE_C_8100_TAG	Yes	DNX_PP_ETHERNET_FRAME_VLAN_FORMAT_S_PRIORITY_C_TAG
DNX_TPID_VALUE_S_88A8_TAG	DNX_TPID_VALUE_C_8100_TAG	Yes	DNX_PP_ETHERNET_FRAME_VLAN_FORMAT_S_PRIORITY_C_TAG
DNX_TPID_VALUE_S_9100_TAG	DNX_TPID_VALUE_S_9100_TAG	No	DNX_PP_ETHERNET_FRAME_VLAN_FORMAT_S_S_TAG
DNX_TPID_VALUE_S_88A8_TAG	DNX_TPID_VALUE_S_88A8_TAG	No	DNX_PP_ETHERNET_FRAME_VLAN_FORMAT_S_S_TAG
DNX_TPID_VALUE_C_8100_TAG	DNX_TPID_VALUE_C_8100_TAG	No	DNX_PP_ETHERNET_FRAME_VLAN_FORMAT_C_C_TAG

NOTE: The classification is associated with all PP ports, both in the ingress and in the egress, with the default native reserved mappings for ingress and egress.

16.4 VLAN Edit Attributes per AC-LIF

Dedicated AC-LIF attributes are used by the VLAN Edit mechanism. The following attributes are supported:

- VLAN-Edit profile – Used together with the Tag structure to map an a VLAN-Edit command ID. A suggested usage is to set the same VLAN-Edit profile for all the LIFs that are expected to have the same tags (TPID wise) after the VLAN-Edit process.
- New Outer-VID, New Inner-VID – Used by the VLAN-Edit mechanism as part of the options to select the VID value of a constructed tag.

The API supports all InLIF, OutLIF and virtual AC formats, both for Outer Ethernet and Native Ethernet ACs. ARP entries with VLAN Edit information and Extender LIFs are supported as well.

The API is asymmetrical, and each side should be configured separately.

16.4.1 SOC Properties

None

16.4.2 Configuration Flow

To configure VLAN Edit attributes for an AC-LIF, call the following API:

`bcm_vlan_port_translation_set(unit, vlan_port_translation)`

- `vlan_port_translation.flags`
  - `BCM_VLAN_ACTION_SET_INGRESS` – The configuration is applicable for the Ingress side.
  - `BCM_VLAN_ACTION_SET_EGRESS` – The configuration is applicable for the Egress side
- `vlan_port_translation.gport` – The AC object for which the VLAN Edit attributes are configured.
- `vlan_port_translation.new_outer_vlan` – A VID value for a constructed Outer Tag. Can be used also for an Inner tag. Any value in the 4K range is valid.
- `vlan_port_translation.new_inner_vlan` – A VID value for a constructed Inner Tag. Can be used also for an Outer tag. Any value in the 4K range is valid. Invalid for a Virtual Egress Default gport (zero value should be set).
- `vlan_port_translation.vlan_edit_class_id` – The VLAN Edit Profile to be used for VLAN Edit command mapping.

16.4.3 Shell Commands

None

16.4.4 Application Reference

Example of VLAN Edit attribute setting for Egress PWE

- **Type:** CINT reference
- **Path:** cint_vpls_native_eve_test.c

Example of VLAN Edit attribute setting for an ARP+AC OutLIF

- **Type:** CINT reference
- **Path:** cint_ip_route_basic.c

16.5 VLAN Edit Command Mapping

The selected VLAN Edit command ID is a result of a mapping between the LIF VLAN Edit Profile and the classified Tag format.

The API is asymmetrical. Each side should be configured separately.

The API expects the mapped Tag format to be allocated prior to the mapping configuration. The VLAN Edit Command-ID can be allocated prior to the mapping or afterwards.

Post egress-VLAN editing processing is performed according to the resulting outer VLAN tag. The processing can include TX tag removal, VLAN-membership filtering, and outbound mirroring. By default, those actions are enabled and controlled per command mapping. It is the user's responsibility to configure the post-EVE processing.

NOTE: Avoid enabling post-EVE processing on untagged packets. This may produce undetermined results.

16.5.1 SOC Properties

None

16.5.2 Configuration Flow

To map a VLAN Edit Profile and Tag format to a VLAN Edit command ID call the following API:

`bcm_vlan_translate_action_class_set(unit, action_class)`

- action_class.flags
  - `BCM_VLAN_ACTION_SET_INGRESS` – The configuration is applicable for the Ingress side.
  - `BCM_VLAN_ACTION_SET_EGRESS` – The configuration is applicable for the Egress side.
  - `BCM_VLAN_ACTION_SET_EGRESS_DISABLE_OUTER_TAG_REMOVAL` – Disable post-EVE removal of the outer TAG (default value is enabled).
  - `BCM_VLAN_ACTION_SET_EGRESS_DISABLE_MEMBERSHIP` – Disable post-EVE VLAN membership filtering (default value is to enable VLAN membership filtering by the outer VLAN).
  - `BCM_VLAN_ACTION_SET_EGRESS_ENABLE_INNER_MEMBERSHIP` – Enable post-EVE VLAN membership filtering using inner VLAN (default value is enable VLAN membership filtering by outer VLAN).
  - `BCM_VLAN_ACTION_SET_EGRESS_DISABLE_MIRRORING` – Disable post-EVE mirroring (default value is to enable VLAN mirroring using the outer VLAN).
- `action_class.vlan_edit_class_id` – The mapped VLAN-Edit Profile
- `action_class.tag_format_class_id` – The mapped tag format
- `action_class.vlan_translation_action_id` – The mapping result VLAN Edit Command ID

16.5.3 Shell Commands

None

16.5.4 Application Reference

Refer to `src/appl/reference/dnx/appl_ref_vlan_init.c`. See VLAN the translation initialization function, `appl_dnx_vlan_translate_init()`.

At egress:

- Map the allocated Tag formats with the whole range of VLAN Edit Profiles to the pre allocated VLAN Edit Command ID 1 that performs no VLAN Edit manipulation.
- Map the Tag format 0 and VLAN Edit Profile 0 to the pre-allocated VLAN Edit Command ID 1 that adds a tag which VLAN is the outgoing VSI value.

At Ingress, no initial mapping is performed and the default value 0 as the VLAN Edit Command ID, requires only the allocation of the Command ID 0 and setting it to perform no VLAN Edit manipulation.

NOTE: Tag formats and Command IDs 0 and 1 should be allocated at the Egress prior to mapping configuration.

16.6 VLAN Edit Command Setting

A VLAN Edit command ID defines the exact manipulation that is selected for the edited VLAN header. This includes removal of up to two existing tags and creation of up to two new tags. For each created tag the user may select the TPID, VLAN ID and the PCP-DEI value.

A VLAN Edit command ID has to be allocated before it can set.

The API is asymmetrical, and each end should be configured separately.

The most significant attributes of a VLAN Edit command are the combination of actions for the outer and the inner tags. The actions serve a dual purpose by defining both the modified tags composition and the source for each created tag VLAN-ID.

The definition of the modified tags composition dictates the tags that will be removed as well as the tags that will be created and their order. Per Outer/Inner tag the action may mean one of the following operations:

- Add – Add a tag that is prior to both referenced tags.
- Replace – Delete the referenced tag and add another tag at the same relative position
- Delete – Delete the referenced tag
- NOP – No operation

After applying the operations to define the modified tags composition, the source VID notion is taken into account. The VID can come from various sources:

- The incoming packet – One of the two outermost tags
- Configured values per LIF – `new_outer_vlan` or `new_inner_vlan`
- VSI – Only in egress

Whenever the Source-VID is the LIF's attribute `new_outer_vlan` or `new_inner_vlan`, outer and inner will refer to the added tags after the tags modification, so the outermost added tag will use the `new_outer_vlan` attribute, while the next tag may use the `new_inner_vlan` attribute.

A constructed tag, whether due to the Add or Replace operation, will reconstruct the TPID and PCP-DEI values as well according to additional input parameters. The TPID should be directly specified and should be one of the configured global TPIDs. The PCP-DEI is constructed according to a specified PCP-DEI action that depicts the source for the PCP-DEI values:

- The incoming packet – One of the two outermost tags
- QoS Mapping – As defined for the specific LIF

Processing the VID-oriented actions is enough in most cases, but the API provides an additional ability to modify the TPID or PCP-DEI value for tags that do not require VID modification. This can be achieved by utilizing the TPID-Action parameters: outer_tpid_action and inner_tpid_action. If the outer or inner tags aren't expected to be modified according to the VID actions, a value other than None in the corresponding TPID-Action implies that the tag should undergo a Replace operation where the VID remains the same, but the TPID and PCP-DEI are reconstructed according to their input parameters.

At Egress only, the TPID maybe overridden by the TPID of one of the outermost tags of the incoming packet. This is done by selecting a TPID-Action that states that either the TPID of outer tag or the inner tag should be selected.

The same Egress VLAN Edit Command configuration is applicable for the Forward stage as well as for any of the supporting Encap stages. At Encap stage it's assumed that the constructed Ethernet header has no tags prior to implementing the VLAN-Edit Command. The Command-ID value is referring to a specific entry in the Forward stage, yet whenever it's mapped by a reset Tag-Format (Untagged), this VLAN Edit Command is synced with Encap stage commands according to the VLAN-Edit Profile value. Validations are performed in order to ensure that commands that are mapped by an Untagged tag format will not include invalid behavior such as deleting tags or using TPID from the incoming packet.

16.6.1 SOC Properties

None

16.6.2 Configuration Flow

To allocate a VLAN Edit command ID, use the following API:

`bcm_vlan_translate_action_id_create(unit, flags, action_id)`

- `flags`
  - `BCM_VLAN_ACTION_SET_INGRESS` – The configuration is applicable for the Ingress side.
  - `BCM_VLAN_ACTION_SET_EGRESS` – The configuration is applicable for the Egress side.
  - `BCM_VLAN_ACTION_SET_WITH_ID` – The user specifies an unallocated VLAN Edit Command ID in the parameter `action_id`.
- `action_id` – The retrieved Command ID and also the user supplied Command ID when used with the flag `BCM_VLAN_ACTION_SET_WITH_ID`.

  To free a non-reserved VLAN Edit Command ID, use `bcm_vlan_translate_action_id_destroy()` and to free all non-reserved Command IDs use `bcm_vlan_translate_action_id_destroy_all()`.

To set an allocated VLAN Edit command ID use the following API:

`bcm_vlan_translate_action_id_set(unit, flags, action_id, action)`

- `flags:`
  - `BCM_VLAN_ACTION_SET_INGRESS` – The configuration is applicable for the Ingress side.
  - `BCM_VLAN_ACTION_SET_EGRESS` – The configuration is applicable for the Egress side.
- `action_id` – The allocated VLAN Edit command ID to be set.
- `action.dt_outer/dt_inner` – The Action to apply on the referenced Outer/Inner. If a tag is to be created, the value also determines where should the VLAN-ID be taken from:
  - `bcmVlanActionNone` – Do not modify the tag
  - `bcmVlanActionAdd` – Add a VLAN tag, where the VLAN-ID is taken from the VLAN attribute of the LIF (For example for `action.dt_outer` the VLAN ID value will be `vlan_port_translation.new_outer_vlan`).
  - `bcmVlanActionReplace` – Delete the referenced tag and replace it with a VLAN-ID that is taken from the VLAN attribute of the LIF (For example for `action.dt_inner` the VLAN ID value will be `vlan_port_translation.new_inner_vlan`).
  - `bcmVlanActionDelete` – Delete the referenced tag.
  - `bcmVlanActionCopy` – Delete the referenced tag and replace it with a VLAN-ID that is taken from the unreferenced incoming packet's tag (For example for `action.dt_outer` the VLAN ID value will be VID from the inner tag).
  - `bcmVlanActionMappedAdd` – Add a VLAN tag, where the VLAN-ID is the Out-VSI. Applicable only for Egress.
  - `bcmVlanActionMappedReplace` – Delete the referenced tag and replace it with a VLAN tag, where the VLAN-ID is the Out-VSI. Applicable only for Egress.
  - `bcmVlanActionOuterAdd` – Add a VLAN tag, where the VLAN-ID is taken from the Outer tag of the incoming packet.
  - `bcmVlanActionInnerAdd` – Add a VLAN tag, where the VLAN-ID is taken from the Inner tag of the incoming packet.

  - `bcmVlanActionArpVlanTranslateAdd` – Add a VLAN tag, where the VLAN-ID is taken from the VLAN attribute of the ARP + AC LIF (For example, for `action.dt_outer`, the VLAN ID value will be `vlan_port_translation.new_outer_vlan`).
- `action.dt_outer_pkt_prio/action.dt_inner_pkt_prio` – Determine where the PCP-DEI value is to be taken for the constructed Outer tag, if applicable.
  - `bcmVlanActionNone` – No PCP-DEI change from the referenced Outer/Inner tag. Use the same PCP-DEI value.

    NOTE: `bcmVlanActionNone` should be used only for the incoming tagged packet. For an untagged packet, `bcmVlanActionNone` is an invalid configuration, as this action will always fetch the PCP-DEI value *from the packet*.
  - `bcmVlanActionAdd`/`bcmVlanActionReplace` – Use the QoS mapped value according to the referenced Outer or Inner tag.
  - `bcmVlanActionCopy` – Get the PCP-DEI from the unreferenced incoming packet's tag (For example for `action.dt_outer_pkt_prio` use the PCP-DEI of the inner tag).
  - `bcmVlanActionOuterAdd` – Get the PCP-DEI from the Outer tag of the incoming packet.
  - `bcmVlanActionInnerAdd` – Get the PCP-DEI from the Inner tag of the incoming packet.
- `action.outer_tpid_action/inner_tpid_action` – Define action that is not VID related.
  - `bcmVlanTpidActionNone` – No additional action.
  - `bcmVlanTpidActionModify` – The refernced tag to perform a Replace even if no such action is required due to action.dt_outer/dt_inner processing. In such case the VID won't change.
  - `bcmVlanTpidActionInner` – The constructed tag will get the TPID from the incoming packet's Inner tag instead of the TPID from `action.outer_tpid/inner_tpid.` Same functionality as bcmVlanTpidActionModify in determining an additional Replace operation.
  - `bcmVlanTpidActionOuter` – The constructed tag will get the TPID from the incoming packet's Outer tag instead of the TPID from `action.outer_tpid/inner_tpid.` Same functionality as bcmVlanTpidActionModify in determining an additional Replace operation.
- `action.outer_tpid/inner_tpid` – One of the configured global TPIDs to be set when constructing the referenced tag or 0 when not applicable.

16.6.3 Shell Commands

None

16.6.4 Application Reference

Refer to `src/appl/reference/dnx/appl_ref_vlan_init.c`. See the VLAN translation initialization function, `appl_dnx_vlan_translate_init()`.

At ingress:

- Create Action-ID 0 and set it to perform no modification of the packet. By default, all the Ingress VLAN-Edit Command mappings are mapped to this value.

At egress:

- Create Action-ID 1 and set it to perform no modification of the packet. In addition, set all the Egress VLAN-Edit Command mappings to this value.
- Create Action-ID 0 and set it to add a tag with a VID that equals to the Out-VSI and TPID 0x8100. This is a typical default for the Routing application. Map the Untagged packets to this Action-ID.

16.7 API Descriptions

16.7.1 VLAN Tag Format Classification

API Name	Highlights
`bcm_port_tpid_class_create(int unit, uint32 flags, bcm_port_tag_struct_type_t tag_struct_type, bcm_port_tag_format_class_t *tag_format_class_id);`	Allocate a tag format for classification
`bcm_port_tpid_class_destroy(int unit, uint32 flags, bcm_port_tag_format_class_t tag_format_class_id)`	Free a tag format for classification
`bcm_port_tpid_class_set(int unit, bcm_port_tpid_class_t *tpid_class)`	Set a tag format classification per port and parser output
`bcm_port_tpid_class_get(int unit, bcm_port_tpid_class_t *tpid_class)`	Retrieve a tag format classification per port and parser output

16.7.2 VLAN Edit Attributes per AC-LIF

API Name	Highlights
`bcm_vlan_port_translation_set(int unit, bcm_vlan_port_translation_t *vlan_port_translation)`	Set VLAN-Edit attributes per AC-LIF.
`bcm_vlan_port_translation_get(int unit, bcm_vlan_port_translation_t *vlan_port_translation)`	Retrieve VLAN-Edit attributes per AC-LIF.

16.7.3 VLAN Edit Command Mapping

API name	Highlights
`bcm_vlan_translate_action_class_set(int unit, bcm_vlan_translate_action_class_t * action_class)`	Sets a VLAN translation mapping that maps a VLAN Edit Profile and Tag format to a VLAN Edit command ID.
`bcm_vlan_translate_action_class_get(int unit, bcm_vlan_translate_action_class_t * action_class`	Gets a VLAN translation mapping that retrieves the VLAN Edit command ID to which the VLAN Edit Profile and Tag format are mapped.

16.7.4 VLAN Edit Command Setting

API Name	Highlights
`bcm_vlan_translate_action_id_create( int unit, uint32 flags, int *action_id)`	Allocate a VLAN-Edit Action-ID
`bcm_vlan_translate_action_id_destroy( int unit, uint32 flags, int action_id, bcm_vlan_action_set_t action)`	Free a VLAN-Edit Action-ID, Configure the VLAN-Edit Action
`bcm_vlan_translate_action_id_destroy_all( inta unit, uint32 flags)`	Free all the VLAN-Edit Action-IDs in a device.
`bcm_vlan_translate_action_id_set( int unit, uint32 flags, int action_id, bcm_vlan_action_set_t *action)`	Configure the VLAN-Edit Action
`bcm_vlan_translate_action_id_get( int unit, uint32 flags, int action_id, bcm_vlan_action_set_t *action)`	Retrieve a VLAN-Edit Action

Chapter 17: Standardized Recycle Header

17.1 Introduction

Recycle (RCY) is a process where the outgoing port of a packet is a port on the RCY interface, and thus the outgoing packet enters a new ingress process, where the incoming port of the second pass is the outgoing port of the first pass.

There are no packet-processing limitations on the RCY flow, but this document discusses some well-defined RCY flows used in some of the PP applications.

Those flows are a Recycled-Header (RCH) based flows, which means that the packet is expected to be encapsulated with an RCH at the end of the first Egress pass (instead of an ETH header), to pass the relevant infiltration from the first pass to the second PP pass. The Recycle-Header has a well-defined structure - Standardized Recycle Header, and this document also reviews its structure and the usage of each of its fields in each RCY flow.

NOTE: Two formats of RCH are described in this chapter. The first format is called standard RCH (RCH), and it is used in most of the flows. The second format is called RCH_SRv6_USP_PSP, and it is used for some SRv6-related flows, but not all of them.

17.2 Recycle Port

17.2.1 SOC Properties

To allocate a recycle port for RCH header handling, the port interface must be a recycle interface, and its IN header type must be RCH_0, RCH_1, or RCH_SRv6_USP_PSP. For example, port 40 will be the dedicate recycle port by calling:

- `ucode_port_40=RCY.0`
- `tm_port_header_type_in_40=RCH`

For more information on port configuration, see Section 2.1, Packet Processing Ports.

17.2.2 Configuration Flow

To configure the recycle port's header-type, call `bcm_switch_control_indexed_port_set(unit, port, key, value)`.

- Incoming header type:
  - `port` – Local-Recycle-Port
  - `key.type` – bcmSwitchPortHeaderType
  - `key.index` – 1 (incoming)
  - `value.value`
    - `BCM_SWITCH_PORT_HEADER_TYPE_RCH_0`
    - `BCM_SWITCH_PORT_HEADER_TYPE_RCH_1`
    - `BCM_SWITCH_PORT_HEADER_TYPE_RCH_SRV6_USP_PSP`
- Outgoing header type:
  - `key.type` – `bcmSwitchPortHeaderType`
  - `key.index` – 2 (outgoing)
  - `value.value` – `BCM_SWITCH_PORT_HEADER_TYPE_ETH`

Some recycle applications require recycle port configuration and cannot coexist on the same port. This is done by calling `bcm_port_control_set(unit, port, bcmPortControlRecycleApp,value)`

- `port` – Configure the in recycle port
- `value` – The following options exist:
  - `bcmPortControlRecycleAppDefault` – Default, used in case applications do not need port attributes.
  - `bcmPortControlRecycleAppDropAndContinue` – Drop & Continue.
  - `bcmPortControlRecycleAppExtendedTerm` – Extended-Termination.
  - `bcmPortControlRecycleAppExtendedEncap` – Extended-Encapsulation, when forwarding decision is done in the first pass.
  - `bcmPortControlRecycleAppExtendedEncapUncollapse` –Extended-Encapsulation, when forwarding decision is done in the second pass.
  - `bcmPortControlRecycleAppSbfdReflector` – SBFD Reflector

NOTE: Configuring a port to be `bcmPortControlRecycleAppDropAndContinue` is the same as setting `bcmPortControlOverlayRecycle` to 1 (enable).

For RCH_1, to derive the PP port from source-system-port of the first pass incoming-port, use the following API to call for each such source-system-port:

```
bcm_port_control_set(unit, port, bcmPortControlSystemPortInjectedMap, enable)
```

- `port` – Local-port of the first pass incoming-port
- `enable` – 1 or 0.

17.2.3 Shell Commands

None

17.3 Control AC-LIF

The Control AC-LIF feature is used in two-pass applications. It has a local LIF that results in recycle header termination in the second pass. The main purpose of Control AC-LIF is to retrieve AC or pipe information from the first pass, like QoS, TTL properties, and others. Not all flows use the control-LIF option on RCH; therefore, the relevant field is called CTRL-LIF or Flow-ID.

17.3.1 SOC Properties

None

17.3.2 Configuration Flow

Create a Control AC LIF by calling `bcm_vlan_port_create(unit, vlan_port)`.

- `vlan_port.flags` – `BCM_VLAN_PORT_CREATE_INGRESS_ONLY` and `BCM_VLAN_PORT_RECYCLE`.
  The `BCM_VLAN_PORT_CREATE_INGRESS_ONLY` and `BCM_VLAN_PORT_WITH_ID` flags can be also used by the user with the `BCM_VLAN_PORT_RECYCLE` flag.
- `vlan_port.criteria` – `BCM_VLAN_PORT_MATCH_NONE`.
- `vlan_port.vsi` – 0.
- `vlan_port.port` – 0.
- QOS parameters: Uniform or Short-pipe.

To connect a Control AC LIF to an Encapsulate Recycle Entry, see Section 17.4, Encapsulation Recycle Entry.

17.3.3 Shell Commands

None

17.3.4 Application Reference

Example of control AC-LIF in VXLAN Bud Node configuration:

- **Type**: CINT reference
- **Path**: `$SDK/src/examples/dnx/cint_vxlan_bud_node.c`

17.4 Encapsulation Recycle Entry

The section explains how to create recycle entries at the egress encapsulation stage using OutLIF object. The encapsulation RCY entry holds the information that is user-configurable over the RCH. It also identifies the RCH type.

17.4.1 SOC Properties

None

17.4.2 Configuration Flow

To configure the recycle entry, call the API `bcm_l2_egress_create(unit, &egr)`

- `egr.flags`: The flags in the following table are supported:

Supported Flags	Description
`BCM_L2_EGRESS_RECYCLE_HEADER`	Mandatory flag to use. Indicates recycle header construction.
`BCM_L2_EGRESS_REPLACE`	Replace an already allocated recycle entry. The flag can be used only with the `BCM_L2_EGRESS_WITH_ID` flag and a valid `encap_id`.
`BCM_L2_EGRESS_WITH_ID`	Create a recycle entry with ID passed by `egr > encap_id`.
`BCM_L2_EGRESS_DEST_PORT`	Set Destination-port as part of Cross-Connect information for Extended-Encapsulation application.
`BCM_L2_EGRESS_EXTENDED_COPY_DEST_ENCAP`	Indicates the first-stack OutLIF from RCH should be copied to the next pass OutLIFs stack. This flag is valid for the Extended-Encapsulation application.

- `egr.encap_id` – `encap_id` object value, returned after the structure configuration. Used later by multicast group copy.
- `egr.vlan_port_id` – Configure the recycle Control InLIF on the RCH header. If not configured, the RCH header will be configured with a default recycled Control InLIF. For more information, see Section 17.3, Control AC-LIF. For a value of 0, the default configuration is used. PHB, remarking, ECN, and TTL Propagation models for default recycle InLIF are uniform.

  This `vlan_port` is not used in the extended encapsulation flow.
- `egr.recycle_app` – The following options exist:
  - `bcmL2EgressRecycleAppDropAndContinue` – Drop-and-Continue application or default value in case application does not need RCH header indication.
  - `bcmL2EgressRecycleAppExtendedTerm` – Extended-Termination. In that case, `recycle_header_extension_length` and `additional_egress_termination_size` can be used.
  - `bcmL2EgressRecycleAppExtendedEncap` – Extended-Encapsulation. Use this option for both regular and uncollapsed models. In that case, `BCM_L2_EGRESS_DEST_ENCAP_ID`, `BCM_L2_EGRESS_DEST_PORT`, `dest_port`, `dest_encap_id`

  - `bcmL2EgressRecycleAppRedirectVrf` – This option is for an L3 application doing two-pass processing where the VRF is configured in the RCH header (instead of the VRF coming from the ingress pipeline).
  - `bcmL2EgressRecycleAppSrv6UspPsp` – SRv6 PSP USP application. Defines an RCH_SRv6_USP_PSP encapsulation.
- `egr.recycle_header_extension_length` – Specify in bytes the size of extension header after recycle header
- `egr.additional_egress_termination_size` – Specify in bytes the size of egress additional termination
- `egr.dest_port, egr.dest_encap_id` – Cross-Connect information for the extended-Encapsulation app.
- `egr.vrf` – Specify VRF for Redirect VRF application.
- `egr.egress_qos_model` – Specify whether the QoS model for `bcmL2EgressRecycleAppExtendedEncap` app. nwk_qos of the recycle header is copied from the system-header if egress_qos type is initial, or inherited from the previous stage if egress_qos type is uniform. The egress_ttl and egress_ecn options are useless.

17.4.3 Shell Commands

None

17.4.4 Application Reference

Example of Recycle encapsulation entry for SRv6 USP PSP:

- **Type:** CINT reference
- **Path:** `$SDK/src/examples/dnx/srv6/cint_srv6_basic.c`

Example of Recycle encapsulation entry for drop and continue:

- **Type:** CINT reference
- **Path:** `$SDK/src/examples/dnx/utility/cint_dnx_util_rch.c`

Example of Recycle encapsulation entry for VRF redirect:

- **Type:** CINT reference
- **Path:** `$SDK/src/examples/dnx/cint_dnx_ip_route_vrf_redirect.c`

Example of Recycle encapsulation entry for extended encapsulation:

- **Type:** CINT reference
- **Path:** `$SDK/src/examples/dnx/srv6/cint_srv6_ext_encap_tunnel.c`

Example of Recycle encapsulation entry for extended termination:

- **Type:** CINT reference
- **Path:** `$SDK/src/examples/dnx/utility/cint_dnx_util_srv6.c`

17.5 RCH Flows

17.5.1 Drop and Continue

For details, see Chapter 18, Drop-and-Continue.

- Recycle port: `bcmPortControlRecycleAppDropAndContinue`.
- Recycle encapsulation entry: `bcmL2EgressRecycleAppDropAndContinue`.
- The RCY process is as follows:
  - Egress first pass: No PP processing at first pass (except for termination of the terminated headers) and RCH encapsulation.
  - Ingress second pass: Use dedicated DBs, which are marked as 2ND_pass DBs.

17.5.2 Extended Termination

The main use case is SRv6. For details, see Chapter 41, Segment Routing over IPv6.

- Recycle port: `bcmPortControlRecycleAppExtendedTerm`.
- Recycle encapsulation entry: `bcmL2EgressRecycleAppExtendedTerm`.
- The RCY process is as follows:
  - Egress first pass: No PP processing at first pass (except for termination the terminated headers) and RCH encapsulation.
  - Ingress second pass:
    - RCH termination, CTRL-LIF is not used.
    - InLIF reclassification.
    - Continue as usual afterwards .

17.5.3 Extended Encapsulation

The main use case is SRv6 extended encapsulation. For details, see Chapter 41, Segment Routing over IPv6.

- Recycle port: `bcmPortControlRecycleAppExtendedEncap`.
- Recycle encapsulation entry: `bcmL2EgressRecycleAppExtendedEncap`.
- The RCY process is as follows:
  - Egress first pass: Regular egress process for termination, forward, and encapsulation, except for ETH and AC encapsulation (use RCH encapsulation instead).
  - Ingress second pass: RCH termination. No further processing until PMF.

17.5.4 Extended Encapsulation – Uncollapse (EEU)

The main use case is EVPN BUM with RCY.

- Recycle port: `bcmPortControlRecycleAppExtendedEncapUncollapse`
- Recycle encapsulation entry: `bcmL2EgressRecycleAppExtendedEncap`
- The RCY process is as follows:
  - Egress first pass: Regular egress process for termination, forward, and encapsulation, except for ETH and AC encapsulation (use the RCH header instead).
  - Ingress second pass: RCH termination with regular ingress processing after that.

17.5.5 VRF Redirect

The main use case is to update the ingress VRF using the RCH header. The VRF update is part of a policy-based routing (PBR) technique that updates only the VRF.

- Recycle port: `bcmPortControlRecycleAppDropAndContinue`
- Recycle encapsulation entry: `bcmL2EgressRecycleAppRedirectVrf`
- The RCY process is as follows:
  - Egress first pass: No egress processing on the first pass (except for termination of the terminated headers) and RCH encapsulation.
  - Ingress second pass: VRF/VSI is updated according to the RCH header. Continue from there.

17.5.6 SBFD Reflector

For information about the SBFD reflector endpoint and SOC property configuration, see Chapter 34, BFD.

17.6 API Descriptions

17.6.1 Recycle Entry

API Name	Highlights
`bcm_l2_egress_create()`	Create recycle entry and set its properties.
`bcm_l2_egress_destroy()`	Destroy recycle entry given recycle encap-ID.
`bcm_l2_egress_get()`	Retrieve recycle entry properties given recycle encap-ID.

17.6.2 Recycle Port

API Name	Highlights
`bcm_port_control_set()`	Create recycle port and set its properties.
`bcm_port_control_get()`	Get recycle port indication

Chapter 18: Drop-and-Continue

18.1 Introduction

Drop-and-continue is a general flow that implements decoupled multilayer forwarding, such as simultaneous forwarding of copies of a received multicast packet on the multicast tree (*continue*), and sending to a locally attached network (*drop*).

Multicast tree nodes are classified as three node types:

- **Trunk node**: Connected to the tree by more than a single link. A trunk node receives and transmits packets on the overlay network. This node type is not connected to the local transmitter/recipient.
- **Leaf node**: Receives packets from the overlay network, terminates overlay headers, and forwards to local copies. The leaf node receives packets from the local network, encapsulates labels, and forwards to the overlay links. Leaf nodes are connected to the tree by a single link and are connected to many local transmitters/recipients
- **Bud node**: Attached to both the overlay network and to the local network. The packet received from the overlay is both on the MPLS overlay network and the local copies. The packet to local terminates their MPLS stack, exposes the local network header (typically IP) and to a second forwarding stage according to it. The bud node also receives a packet from the local network, encapsulates labels, and forwards to the overlay links in a single pass as the Leaf node.

When a bud node receives traffic on the MC tree, the bud node processes the packet using the drop-and-continue flow:

- One or more copies *continue* on the MC overlay tree.
- One copy may be *dropped* and transmitted to the local recipient(s)

NOTE: The MC traffic initiated on a bud node applies the regular MC flow (that is, single pass).

The SDK supports drop-and-continue flows of MLDP, and VXLAN applications.

Before reading this chapter, make sure you are familiar with the concepts in the following chapters:

- Chapter 21, Ethernet Bridge and Chapter 22, IP Router (basic L2 and L3 object overview)
- Chapter 28, IP Tunnel v4 Termination, Chapter 30, IP Tunnel v6 Termination, and Chapter 32, VXLAN IP Overlay (details about the tunnel APIs and the additional possible configurations)
- Chapter 24, MPLS LER Termination (details on MPLS Termination APIs and possible MPLS configurations)

18.2 Configuration Checklist

- Allocate recycle port for all packets that use RCH header
- First pass:
  - Add Encapsulation Recycle entries.
  - VXLAN Bud-Node: Configure IPMC flow including L3 port properties, MyMac per ETH-RIF and L3 IPMC entries for multicast copies. See more information on IPMC flow in Chapter 22, IP Router
  - MPLS Bud-Node (MLDP): Configure MPLS MC flow including L3 port properties, MyMac per ETH-RIF and MPLS LSR entries for multicast copies. See more information on MPLS LSR flow in Chapter 23, MPLS Label Switch Router.
  - One of the copies in multicast group should go to Recycle port and Recycle entry.
- Second pass:
  - Configure Bud entries of VXLAN tunnel terminator entry or MPLS Tunnel termination to terminate second pass.
  - Configure L2 MACT UC Forwarding or L2 MC entry for copies going to the local recipients on the second pass for VXLAN Overlay. For more information, see Chapter 21, Ethernet Bridge.
  - Configure forwarding entry, according to the overlay network.

- Configure tunnel termination object on the second pass. For IP or VXLAN tunnel termination call `bcm_tunnel_terminator_create(unit, tunnel)` with the flag `BCM_TUNNEL_TERM_BUD`. To configure MPLS tunnel termination on the second pass call `bcm_mpls_tunnel_switch_create (unit, info)` with the flag `BCM_MPLS_SWITCH2_TERM_BUD`.

18.3 Configuring Second-Pass Tunnel Termination

In the bud node, the MPLS multicast label or IP tunnel termination occurs on the second pass. The packet arrives on the recycling interface with a dummy Ethernet header and the MPLS multicast label or labels as the outermost labels or IP tunnel header. The dummy Ethernet header is terminated at VTA, and the MPLS multicast label is looked up in a second-pass local-multicast database (different from the one used in the first pass), which resides in either the SEMs or the TCAM DB. The result of the second pass multicast termination is an LSP or PWE-MP (according to the label type). The MPLS termination process itself (i.e., capabilities in terms of MPLS stack depth, TTL inheritance, special label identification, and so on) is identical to the first-pass MPLS label termination process and procedures. IP tunnel termination—both IPv4 and IPv6, with or without additional headers—supports the same DBs and lookup keys as in tunnel termination on the first pass.

18.3.1 SOC Properties

None

18.3.2 Configuration Flow

To configure the IP tunnel terminator object on the second pass call API `bcm_tunnel_terminator_create(unit, tunnel)`.

- `tunnel.flags: BCM_TUNNEL_TERM_BUD` – The flag indicates that the lookup information will be stored in tunnel termination DBs, accessed on the second pass.

For all other possible configurations and lookup keys, see the relevant tunnel terminator section.

To configure MPLS tunnel terminator on the second pass call API `bcm_mpls_tunnel_switch_create(unit, info)`

- `info.flags: BCM_MPLS_SWITCH2_TERM_BUD` – The flag indicates that the lookup information will be stored in mLDP label termination DBs.

For all other possible configurations on MPLS ingress termination objects, see Chapter 24, MPLS LER Termination.

18.3.3 Shell Commands

None

18.3.4 Application Reference

Example of VXLAN Bud Node configuration

- **Type:** CINT reference
- **Path:** `$SDK/src/examples/dnx/cint_vxlan_bud_node.c`

Example of MPLS MLDP configuration

- **Type:** CINT reference
- **Path:** `$SDK/src/examples/dnx/mpls/cint_dnx_mpls_mldp.c`

Chapter 19: ROO (Routing Over Overlay)

19.1 Introduction

IP Routing-Over-Overlay (ROO) refers to a set of protocols/applications where the L2 forwarding to the host/next-hop router is not accomplished by simple 802.1q bridging, but by L2-overlay protocols: VXLAN, EVPN or VPLS. This chapter describes the packet processing steps within the device performing routing Routing-Over-Overlay from In-VSI L2 domain to Out-VSI L2 domain.

It is recommended to read this chapter after reading the following sections:

- Chapter 21, Ethernet Bridge and Chapter 22, IP Router for an overview of the basic L2 and L3 objects.
- Section 22.6, ETH-RIF (L3 VSI Interface) and VRF Definition. This section provides information about configurations that are related to next hop information, namely 3-Hierarchy FECs and Egress-Native-ARP. Also, it provides information relevant for the MyMac operation which is a prerequisite for overlay tunnel termination and IP forwarding.
- Tunnel related sections, giving more details on the tunnel APIs and the additional possible configurations: Chapter 32, VXLAN IP Overlay, and the MPLS, VPLS and EVPN chapters.

BCM88690 supports the processing of in-overlay and out-of-overlay networks of the following packets stacks:

- IPo(ETH-native)oVPLSoMPLSoETH(overlay)
- IPo(ETH-native)oVXLANoUDPoIPoETH(overlay)
- IPo(ETH-native)oEVPN/IMLoMPLSoETH(overlay)

NOTE: ESI encapsulation is not supported for ROO.

The outermost Ethernet header is referred to as the *Overlay-Link-Layer* (LL) header or *Overlay-ARP*. The inner Ethernet header is referred to as the *Native-Link-Layer* (LL) or *Native-ARP*.

In Routing over VXLAN, the outermost IP header, its fields, and its associated objects are referred to as the *Overlay-IP* header (for example, Overlay-RIF, Overlay-DIP, Overlay -SIP, etc), the inner IP header is referred to as the *Native-IP* header (for example, Native-DIP, Native -SIP, Native-RIF, and so on).

The packet processing steps within the device performing routing Routing-Over-Overlay from In-VSI L2 domain to Out-VSI L2 domain for VPLS packets are as follows:

- When the packet arrives from a native router to the overlay edge bridge router, it is referred to as *routing into overlay*.
  - The Native IPoETH headers are processed as a regular routing case. On the Native IP forwarding lookup is performed to resolve the destination. In this case it is done by hierarchy FECs/ECMPs.
    - The first FEC could be resolved to the native Out-RIF and native ARP entry. The Native Eth is built based on the information derived from the EEDB, accessed with the Native Out-RIF and ARP pointers. EVE might be performed.The destination is the second FEC.
    - The second FEC is resolved to another OutLIF pointer, usually the OutLIF overlay (Tunnel-LIF). According to the Overlay OutLIF the overlay tunnel is built and mapped to the overlay ARP. From the overlay ARP, the overlay LL entry is built.

NOTE: A three FEC Hierarchy is also possible. This is usually utilized in an Overlay hierarchy configuration, for example providing both VPLS and MPLS encapsulation tunnels using two different FECs. Another example is VXLAN and ARP using two different FECs.

- When a packet arrives from the overlay edge bridge router and its destination is the Native router, it is referred to as *routing out of overlay*.
  - The Overlay ETH is looked up with the DMAC, matching the MyMac of the Overlay router. As a result, the outer Ethernet is terminated.
  - The Overlay-header (IP tunnel/ PWE + MPLS label) lookup is performed according to the overlay network.
  - The last VT stage is dedicated to a second MyMac lookup and termination. The VSI, ETH-RIF from the terminated L2 tunnel together with the DMAC in the inner ETH results in a termination of the inner ETH. The ETH-RIF table together with the DIP from the inner IP header serves as a key to perform regular IP processing.

NOTE: The forwarding lookup might be IP UC and MC. Each replication in the MC group might point to a different Out-RIF and overlay OutLIF.

19.2 Application Configuration Checklist

- Set L3 port properties. For more information, see Section 22.3, Port Routing Properties
- Set ETH-RIF properties and create MyMac per ETH-RIF. For more information, see Section 22.6, ETH-RIF (L3 VSI Interface) and VRF Definition.
- Set L3 Egress object: Ingress-FEC. For more information, see Chapter 22, IP Router.
- Configure Next Hop Information for encapsulating Native and Overlay LL. For more information, see Chapter 22, IP Router.
- Configure L3 Forwarding entry - host/ route. For more information, see Chapter 22, IP Router.
- Configure IP Multicast Forwarding entries and MC groups. For more information, see Chapter 22, IP Router
- Configure MPLS port and ingress/ egress tunnels. For more information, see Chapter 23, MPLS Label Switch Router. This section provides more details on MPLS APIs and possible MPLS configurations.
- Configure VXLAN L2-VPN, ingress/egress tunnel object and VXLAN port.
- Configure EVPN and IML ingress/egress tunnels. For more information, see Chapter 35, Ethernet VPN (EVPN).

19.3 Native L3 Egress Object: Egress-ARP (Next-Hop)

Routing Over Overlay requires two separate ARP objects to be created at egress: one for LL encapsulation of the Native and a second for the overlay header. For more detailed information on ARP creation and additional possible configurations, see Chapter 22, IP Router. This chapter covers only the native ARP creation.

19.3.1 SOC Properties

None

19.3.2 Configuration Flow

- To create an egress object – Egress Natitve ARP, call `bcm_l3_egress_create(unit, allocation_flags, &l3eg, NULL)`
  - l3eg.flags:
    - `BCM_L3_NATIVE_ENCAP` – ARP is required for Native Ethernet layer.

NOTE: For additional API configuration settings, refer to Section 22.7, L3 Egress Object: Egress-ARP (Next-Hop).

19.3.3 Shell Commands

None

19.4 Native L3 Egress Object: Ingress-FEC

Routing over overlay (ROO) requires 2/3 separate FEC objects with different hierarchies to be created at ingress: one for pointing to the Native ARP, one for the next hierarchy FEC, and another for the overlay tunnel header. For more detailed information on FEC creation, additional possible configurations, and ID allocation ranges, see Chapter 22, IP Router.

19.5 ROO with IPV6 Headers

NOTE: The information in this section applies to the BCM88690 and BCM88800 devices only.

ROO with an overlay or native IPV6 header may exceed the supported packet header size, and as a consequence, the packet is not built correctly.

A field processor solution is available in a CINT example.

19.5.1 Application Reference

Example of IPv6 VXLAN ROO configurations:

- **Path:** `$SDK/src/examples/dnx/field/cint_field_etpp_fwd_term_min_bytes.c`
  Main function: `cint_field_etpp_fwd_term_min_bytes_main`
- **Path**: `$SDK/src/examples/dnx/vxlan/cint_vxlan_roo_basic.c`
  Main function: `vxlan_roo_basic() with parameter is_ipv6_vxlan=1`

Chapter 20: L2 Generalized Bridging Model

20.1 Introduction

This chapter describes two L2 interface concepts: network groups, cross-connection and gport forward information .

20.2 Split Horizon (Network Groups)

InLIFs and OutLIFs have an Orientation attribute. At the egress, it is possible to configure an Ethernet forwarding filter on {InLIF Orientation, OutLIF Orientation} to decide whether to forward or filter an Ethernet-forwarded packet. A typical usage of the Orientation attribute is in VPLS, where the Orientations are hub (network core-facing LIFs, typically PWE-LIFs) and spoke (customer-facing LIFs, typically AC-LIFs). Split horizon filters packets coming from the core on Hub-LIFs from being forwarded back to the core, while admitting hub-to-spoke and spoke-to-spoke forwarding. Orientation attributes can also be used in E-Line, where packets coming from the core to leaf-nodes are filtered from being forwarded back to the core, while packets coming from the core to root-nodes are filtered from being forwarded back to the core.

Split horizon filtering is done in the ETPP stage and is fully user-configurable through SDK APIs. By default, only packets that have both incoming and outgoing network group ID 1 are filtered.

The following table summarizes the relevant APIs that use network groups.

Module	API Name	Direction	Field
VLAN PORT	`bcm_vlan_port_create`	Ingress	`ingress_network_group_id`
		Egress	`egress_network_group_id`
MPLS PORT	`bcm_mpls_port_create`	Ingress	`network_group_id`
		Egress	`network_group_id`
Tunnel Gports for L2 applications, such as VXLAN and EVPN	`bcm_port_class_set`	Ingress	`bcmPortClassForwardIngress`
		Egress	`bcmPortClassForwardEgress`
Port (not supported in BCM88690)	`bcm_port_class_set`	Egress	`bcmPortClassForwardEgress`

The orientation per-port is used only if no other LIF orientation attribute is available on the egress stack (not supported in the BCM88690).

20.2.1 Application Configuration Checklist

- Set the desired number of orientation bits using SoC properties
- Define orientation-based filtering (split-horizon) between two orientations.
- Enable the split horizon filter per port.
- Configure the split horizon trap.
- Create the ingress and egress LIF configuration using LIF APIs. The `network_group` parameter in each API represents the orientation value, and the range is defined by the SoC property `in_lif_profile_egress_allocate_orientation` in ingress.
- For tunnels, use `bcm_port_class_set` to configure split horizon groups.
- For L2VPN tunnels (PWE, VXLAN) object, additional Native-AC configuration is required

20.2.2 SOC Properties

`in_lif_profile_egress_allocate_orientation`

20.2.3 Define Orientation-Based Filtering

L2 logical interface APIs that support split horizon filtering (aka `bcm_vlan_port_create`, `bcm_mpls_port_add`, `bcm_vxlan_port_add`) can be defined with `network_group` property. The orientation filter between the different groups can be configured per ingress-egress pair.

20.2.4 Configure Split Horizon for Egress L2 VPN Tunnels (PWE, VXLAN, EVPN)

NOTE: The following functionality is relevant only to the BCM88690 device.

Split horizon for PWE and VXLAN objects should usually be inherited from the PWE and VXLAN tunnels. For the BCM88690, egress split horizon information is inherited from a native AC entry. When this occurs, additional configuration of the native AC is required to make split horizon functional.

- Per Outgoing Network Group ID that is required to be used by Egress L2 VPN Tunnels (PWE, VXLAN) , create a default Native-AC.
- Connect Egress L2 VPN Tunnel with the Native-AC according to the outgoing network group ID.
- Note: The default Native-AC entry handles the outgoing network group ID 0. If no connection is done, the network group ID is 0.

20.2.5 Configuration Flow

1. To configure Orientation value for L2 Tunnels, see table, above. Relating to L3 Tunnels (for example, VXLAN), call:
   `bcm_port_class_set(unit, port, pclass, class_id);`
   where:
   - `port` – Tunnel gport
   - `pclass` – `bcmPortClassForwardIngress` for ingress tunnel
   - `pclass` – `bcmPortClassForwardEgress` for egress tunnel
   - `class_id` – Orientation group

2. To define Orientation-based filtering (Split-Horizon) between two orientations: Call: `bcm_switch_network_group_config_set(unit, source_network_group_id, config)`
   - `source_network_group_id` – The group id of the source, originated from the last terminated LIF.
   - `config.dest_network_group_id` – The destination group id, originated from the first encapsulated LIF.
   - `config.flags`:
     - If `config.flags = BCM_SWITCH_NETWORK_GROUP_EGRESS_PRUNE_ENABLE`, then packets from `source_network_group_id` to `config.dest_network_group_id` will be filtered.
     - If `config.flags = 0`, then packets from `source_network_group_id` to `config.dest_network_group_id` will not be filtered.
3. To enable/disable Split-Horizon per port call: `bcm_port_control_set(unit, port, bcmPortControlForwardNetworkGroup, value)`
   - `port` – For which pp port to enable/ disable the split horizon feature.
   - `value` – Disable/enable (0/1)
4. To define Split Horizon trap, call:
   - `bcm_rx_trap_type_create(unit, flags, bcmRxTrapEgTxUserDefine, &trap_id);`
   - `bcm_rx_trap_config_t_init(&trap_config);`
   - `bcm_rx_trap_set(unit, trap_id, &trap_config);`
5. To set the Trap-Action-Profile for split horizon trap:
   - `BCM_GPORT_TRAP_SET(trap_gport, trap_id, fwd_strength, 0);`
   - `bcm_rx_trap_action_profile_set(unit, app_flags,`
   - `bcmRxTrapEgTxSplitHorizonFilter, trap_gport);`

   For more details on trap configuration see Chapter 12, Traps.

NOTE: The following functionality is required only for the BCM88690.

6. For Egress L2VPN tunnels (PWE, VXLAN), it is required to create Native default ACs for each valid outgoing Network group ID, call bcm_vlan_port_create():
   - `vlan_port.criteria = BCM_VLAN_PORT_MATCH_NONE.`
   - `vlan_port.flags |= BCM_VLAN_PORT_DEFAULT |BCM_VLAN_PORT_NATIVE | BCM_VLAN_PORT_CREATE_EGRESS_ONLY|BCM_VLAN_PORT_VLAN_TRANSLATION.`
   - `vlan_port.egress_network_group_id` – Network group ID.

     This call should be done for every valid outgoing network group id.
7. Once Egress L2VPN tunnel is created, connect the object with the right Native default AC (that match to the right network group ID), call `bcm_port_match_add()`:
   - `port` – Set with `vlan_port.vlan_port_id` (created default AC from previous step).
   - `match_info.match = BCM_PORT_MATCH_PORT.`
   - `match_info.flags |= BCM_PORT_MATCH_NATIVE | BCM_PORT_MATCH_EGRESS_ONLY.`
   - `match_info.port = mpls_port.mpls_port_id` (Egress PWE's gport) or `tunnel_id` (Egress VXLAN-IP Tunnel gport)

20.2.6 Shell Commands

None

20.2.7 Application Reference

Split Horizon configuration example

Example of application AC and PWE ports and configuring split horizon filter on the ports.

- **Type:** Default Application Reference
- **Path:** `$SDK/src/examples/dnx/cint_vpls_split_horizon.c`

20.3 Cross-Connect

The cross-connect application, also known as the point-to-point (P2P) application, enables direct forwarding between AC LIFs and PWE LIFs while skipping the standard MACT lookup for forwarding information.

NOTE: For AC LIFs (VLAN-Ports), at the creation of the object, the AC InLIF is created as a multipoint (MP) type LIF. When cross-connecting a LIF to the destination gport, the InLIF is changed to a P2P InLIF that stores the forwarding information. When dissociating the cross-connection, the InLIF is changed back to its default as an MP-LIF.

20.3.1 Configuration Flow

An egress object might be pointed from a P2P object. There are two *mutually exclusive* ways to configure a P2P scheme using this egress object:

The first way is as follows:

1. Call the API that creates the egress object.
2. Create the ingress object (by calling the same API, with flags indicating it's ingress).
   a. Set the port field: `port == <FEC pointing to the egress object or physical port>`.
   b. Set the `encap_id field: encap_id == <value returned from the egress object's creation>`..
3. Call `bcm_vswitch_cross_connect_add()` to connect between the P2P object to the above created egress object.
   a. Set `gports.flags == BCM_VSWITCH_CROSS_CONNECT_DIRECTIONAL`.
   b. Set `gports.port1 == <Id of P2P object>`.
   c. Set:
      - `gports.port2 == <gport of the egress object created above>`.
      - `gports.encap2 == BCM_FORWARD_ENCAP_ID_INVALID`.
        Or
      - `gports.port2 == <FEC pointing to the egress object or physical port>`.
      - `gports.encap2 == encap_id` (value returned from the egress object's creation).

In the second method, call `bcm_vswitch_cross_connect_add()`

1. Set `gports.port1 == <gport of P2P object>, gports.port2 == <FEC pointing to the egress object or physical port>`.
2. Set `gports.encap_id2 == encap_id` (value returned from the egress object's creation).
3. Set `gports.flags == BCM_VSWITCH_CROSS_CONNECT_DIRECTIONAL`.

NOTE: This sequence can also configure the cross correction for the opposite direction with omitting the `BCM_VSWITCH_CROSS_CONNECT_DIRECTIONAL` flag.

20.3.2 Shell Commands

None

20.3.3 Application Reference

Simple P2P cross-connect scenario

Example of a P2P scenario with AC and PWE objects.

- **Type:** CINT reference.
- **Path:** `src/examples/sand/cint_vswitch_cross_connect_p2p_ac_pwe.c`
- **Main function:** `vswitch_cross_connect_p2p_ac_pwe_with_ports__main_config__start_run()`

Split Horizon example

Example of application AC and PWE ports and configuring split horizon filter on the ports.

- **Type**: Default Application Reference.
- **Path**: `src/examples/dnx/cint_vpls_split_horizon.c`
- **Main function**: `vpls_split_horizon_main()`.

20.4 API Descriptions

20.4.1 Split Horizon

API Name	Highlights
`bcm_switch_network_group_config_set`	Configure split horizon filter between two groups
`bcm_switch_network_group_config_get`	Get split horizon filter configuration between two groups
`bcm_port_class_set`	Configure orientation group of a tunnel (for pclass `bcmPortClassForwardIngress`, `bcmPortClassForwardEgress`)
`bcm_port_control_set`	Configure enable/disable split horizon per port (for type `bcmPortControlForwardNetworkGroup`)
`bcm_port_class_get`	Get orientation group of a tunnel (for pclass `bcmPortClassForwardIngress`, `bcmPortClassForwardEgress`)
`bcm_port_control_get`	Get enable/disable split horizon per port (for type `bcmPortControlForwardNetworkGroup`)

20.4.2 Cross Connect

API Name	Highlights
`bcm_vswitch_cross_connect_add`	Connect two logical ports in a P2P service **NOTE:** For an AC LIF, this API may change an InLIF format from an MP-LIF to a P2P-LIF.
`bcm_vswitch_cross_connect_get`	For a given P2P gport, return its P2P peer.
`bcm_vswitch_cross_connect_delete`	For a given P2P gport, delete the connection to its P2P peer. **NOTE:** For an AC LIF, this API may change an InLIF format from a P2P-LIF to an MP-LIF.

20.5 Gport Forward Information

Forwarding information in DNX can be represented in two ways:

- A combination of physical port/FEC and egress encapsulation pointer (OutLIF).
- A Gport that represents a forwarding object. The mapping of a forwarding object to explicit forward information is stored in the software database during the creation of the object. If ingress and egress objects are created separately, such as with MPLS port APIs, the forwarding object is always the ingress object. In multidevice systems, using the forwarding object gport representation on a certain device is only relevant if this object was previously created on this device.

Forwarding APIs (`bcm_l2_address_add` and `bcm_cross_connect_add`) support both methods of representation upon creation. Get APIs return the gport representation, if it exists. Otherwise, the explicit port and encapsulation ID information is returned by the API.

20.5.1 Configuration Flow

A simple flow example of the forwarding API is as follows:

1. Create an ingress and egress object (`bcm_vlan_port_create`).

   Or, create an egress or ingress object only:
   - Egress object (`bcm_mpls_port_create with BCM_MPLS_PORT_EGRESS_ONLY`)
   - Ingress object (`bcm_mpls_port_create with BCM_MPLS_PORT_INGRESS_ONLY`)

NOTE: The forwarding information is specified in the ingress configuration API.

2. If the system is a multidevice system with no object, sync across devices:
   a. Get forwarding information per gport on the device where the object was created using `bcm_l2_gport_forward_info_get`.
   b. Add L2 forwarding or cross-connect with explicit forwarding information.
3. If the system is not a multidevice system, add L2 forwarding or cross-connect with the gport destination.

20.5.2 Shell Commands

None

20.5.3 API Descriptions

API Name	Highlights
`bcm_l2_gport_forward_info_get()`	Return forwarding information of L2 gport

Chapter 21: Ethernet Bridge

21.1 Introduction

Ethernet bridging is data link layer (Layer 2) forwarding based on MAC addresses and has the following basic properties: Flooding, Forwarding, and Learning.

In this section, the API calls for simple bridging applications are described. Simple bridging refers to (IEEE 802.1q) VLAN Bridge, with some additions that are commonly found in VLAN Bridge implementations.

The application configuration checklist summarizes the main attributes required to configure for the device to meet Ethernet bridging requirements.

Note that VLAN translation configuration (also known as *VLAN editing*) is not in the scope of this chapter and is explained in Chapter 16, VLAN Editing Mechanism. However, this chapter does explain how Port x VLAN objects (AC-LIF) can create VLAN-translation instances.

L2VPN, QinQ, and AC learning are also not covered in this chapter. See Chapter 27, Q-in-Q Bridging.

21.2 Application Configuration Checklist

- TPID Definitions
- Port Bridging Properties
- Port x VLAN Properties
- Define Port x VLAN-Tag-Structure Configuration
- Define VLAN-Translation AC-LIF
- VSI Ethernet Bridging Properties
- MACT Management: Aging
- MACT Management: Flush Machine
- MACT Management: L2 Learn
- MACT Management: Registration Upon MACT Changes
- L2 MACT Forwarding
- STG Properties
- L2 Filters
- In-AC Wide Data
- VLAN Compression
- MACT Independent VLAN Learning (IVL)

21.3 TPID Definitions

TPID (Tag protocol Identifier) allows identifying the 802.1Q-Tagged frame by setting a value of 0x8100. Additional TPIDs can be set and used to identify more VLAN-tag types.

TPIDs need to be defined at the start of the application for the VLAN-tags to be recognized correctly. Each device supports different number of TPIDs which are configured globally. The number of global TPIDs for a specific device can be seen in Appendix A, Device Family Differences.

- TPIDs are configured symmetrically in the device (ingress and egress) and their configuration spans the entire pipeline.
- Each port (incoming and outgoing) can be associated with any of the global TPIDs.
- Incoming-tag-structure is decided according to Port / LIF and the TPIDs on the packet.
- VLAN translation (IVE, EVE commands) use the same global TPIDs namespace when adding new tags.

NOTE: After SDK initialization, no global TPIDs are set. Global TPIDs are expected to be configured as part of application initialization. For an example, see Section 21.3.4, Application Reference. If a global TPID must be configured after initialization, the user is required to disable traffic before performing the command, apart from applying any TPID-related configurations for the modified global TPID setting.

21.3.1 SOC Properties

None

21.3.2 Configuration Flow

To add a new global TPID, call:

```
bcm_switch_tpid_info_t tpid_info;
tpid_info.tpid_value = <TPID_value>;
bcm_switch_tpid_add(unit, 0, &tpid_info);
```

21.3.3 Shell Commands

None

21.3.4 Application Reference

TPID initialization values

Example of application adding global TPIDs to allow traffic for basic VLAN flows. The following TPID values are configured by default: 0x8100, 0x9100, 0x88a8, 0x88e7.

- **Type:** Default Application Reference
- **Path:** `$SDK/src/appl/reference/dnx/appl_ref_vlan_init.c`
- **Function** `appl_dnx_global_tpid_init()`

Compatibility TPID APIs

- **Type:** CINT utility example, shows compatibility between BCM88690 and previous devices.
- **Path:** `$SDK/src/examples/sand/utility/cint_sand_utils_tpid.c`

Example of TPID usage in basic application

- **Type:** CINT reference, shows differences between BCM88690 and previous devices.
- **Path:** `$SDK/src/examples/dnx/cint_dnx_vlan_membership.c`

21.4 Port Bridging Properties

Any Ethernet port has the following bridging properties:

- The port should recognize, accept, and transmit Ethernet packets. For information about configuring a port's header type, see Section 2, Ports and Generalized Ports. The port is considered an Ethernet port and added to the E bitmap in port config if its incoming header type can handle Ethernet header.
- Map a port to a VLAN domain, assuming symmetric configuration between incoming and outgoing sides. A VLAN domain is typically used as part of the key lookup (ISEM, ESEM) to resolve the packet's In-AC and Out-AC or generalization of some bridging properties.
- Initial VID assignment: Assign a packet's VID value according to the incoming port if the packet is untagged. A packet is untagged when the incoming port recognizes the packet's tag structure as untagged. For more information, see Section 21.6, Port x VLAN-Tag-Structure Configuration. In addition, it is possible to ignore the packet VLAN tag on the port and use the initial VID assignment value instead.

NOTE: SDK implies that the configuration of initial VID assignment is per port. Actually the functionality is per VLAN domain.

- Tx-tag VID assignment: Outgoing port can set which VLAN (and only one VLAN) will resolve the packet to be transmitted as untagged even if the VLAN-tag appear on the packet-header.
- VLAN membership interface: Each port is assigned to a VLAN-membership interface. VLAN-membership is checked per VLAN membership interface × VLAN. By default, each port is mapped 1:1 to the VLAN membership interface.
- QoS: Allow PHB and Remark functionality per port. When the QoS model on the incoming port is stronger than those on LIFs, the port QoS is applied for all packets, see Section 10.3, Ingress PHB/Remarking in Chapter 10, QoS.
- Default-LIF (AC) per Port: Port can be used to derive LIF properties for Basic-Bridge/Route if of no match in ISEM (for In-AC) or ESEM (for Out-AC). The configuration of incoming and outgoing sides are done separately.
  - At the ingress, InLIF (In-AC) must be created and incoming port is set to some InLIF ID. LIF-properties are set in the InLIF table. There can be multiple ports mapped to the same InLIF ID.
  - At the egress, outgoing port is set to Out-AC default entry. This is done by having intermediate object (Out-AC Default profile). Multiple outgoing ports can be linked to the same Out-AC Default profile. Out-AC default profiles are exposed to the user.
- MACT SA action can be defined per incoming-port:
  - If MACT SA is not found, one of four trap actions is defined.
- Unknown-DA MACs filtering/action can be defined per incoming-port and outgoing-port. Per outgoing-port action, it is possible only when working in BCM88670 interop mode. For more information see Section 5.5, Port Properties.
- It is possible to change Flooding group per Unknown-UC, Unknown-MC, or Broadcast.
- It is possible to enable or disable same-interface filtering for device scope filtering. The filter is applied by default and complies with the following rule: If the outgoing port equals the incoming port and both interfaces are ports, the packet is filtered. If both interfaces are LIFs, they should also be equal for the device scope filtering purposes.
- It is possible to configure the port to drop all packets arriving or leaving from it with a VLAN tag identifying a VLAN of which the port is not a member.

Additional configurations per port:

- For Port x VLAN properties, see Section 21.6, Port x VLAN-Tag-Structure Configuration.
- For Port x Tag-structure properties, see Section 21.6, Port x VLAN-Tag-Structure Configuration.
- For Port STG properties, see Section 21.15, STG Properties.
- For Port learning properties, see Section 21.12, MACT Management: L2 Learn.

21.4.1 SOC Properties

For port definitions to accept Ethernet packets, see to Traffic Management ports in the *Traffic Manager Theory of Operation* document and Section 2.1, Packet Processing Ports.

21.4.2 Configuration Flow

1. Associate a port with a VLAN domain by calling:
   `bcm_port_class_set(unit, port, bcmPortClassID, vlan_domain)`
   – `port` is the logical physical port (Local-Port/System-Port gport)
   – `bcmPortClassID` stands for VLAN-domain property of a port

NOTE: By default, there is 1:1 mapping between Local-Port and VLAN-Domain. It is expected that VLAN-Domain mapping will be done at the start of port initialization and will not change on run-time.

2. To configure the VLAN membership interface property of a certain port, call `bcm_port_class_set(unit, port, bcmPortClassVlanMember, class_id)`.
   – `port` is the logical physical port (Local-Port or System-Port gport)
   – `class_id` is the VLAN membership interface property of the port.
3. Configure initial-VID assignment:
   – To set untagged packet default VLAN for the incoming-port, call:
   `bcm_port_untagged_vlan_set(unit, incoming_port, vlan)`
   - `vlan` – Packet default VLAN.
     By default, once port is configured to recognize Ethernet packets, SDK set port default VLAN to be 1.
   – Ignore the packet VLAN tag on the port and set the initial-VID as an untagged packet by calling:
   `bcm_vlan_control_port_set(unit, incoming_port, bcmVlanPortIgnorePktTag, mode)`
   - `mode` – Value 0 means do not ignore. Otherwise, ignore.
     By default, the port does not ignore incoming VLAN tag.
4. To set untagged packet default VLAN (TX-tag VID assignment functionality) for outgoing-port, see Section 21.6, Port x VLAN-Tag-Structure Configuration
5. Define the **default LIF** incoming-port. Two options are available:
   – Option 1: On BCM initialization, the SDK creates two default In-ACs.

Type	Description
Initial-AC	All incoming-port traffic assumes VSI equals VLAN if there is no match in ISEM. If the packet is untagged, then VSI equals Initial-VID.
Drop-AC	All incoming-port traffic is assumed to be a drop if there is no match in ISEM.

By default, the SDK assigns all Ethernet ports to Drop-AC. Per port, it is possible to switch between the two default In-ACs by calling

`bcm_vlan_control_port_set(unit, incoming-port, bcmVlanTranslateIngressMissDrop, arg)`

- `arg` – Value 0 means the port is assigned to Initial-AC, otherwise it will be assigned to Drop-AC.

– Option 2: To create a new default In-AC with its defined configuration for incoming-port (or multiple incoming-ports), call `bcm_vlan_port_create` and (optionally) `bcm_port_match_add`. For more information see Section 21.6.4, Application Reference.

NOTE: When Option 2 is selected for a given incoming-port, Option 1 cannot be used.

6. Define the default LIF for the outgoing-port. The user can create an Out-AC default profile (using `bcm_vlan_port_create`) and associate outgoing-ports to it by `bcm_port_match_add`. For more information see Section 21.6.4, Application Reference.

   By default, all outgoing-port traffic go to default Out-AC with the properties of VLAN translation profile 0 (see more information in Section 21.6.4, Application Reference).

7. To configure both port learning enable/disable and set relevant trap action profile for MAC SA not found, call `bcm_port_learn_set(unit, incoming_port, flags)`

   `flags` option:

Flags	Description	Trap-Code Trigger
`BCM_PORT_LEARN_ARL | BCM_PORT_LEARN_FWD`	By default action trap will be learn and forward	`bcmRxTrapL2Learn0`
`BCM_PORT_LEARN_FWD`	By default action trap will be forward and do not learn	`bcmRxTrapL2Learn1`
`BCM_PORT_LEARN_ARL`	by default action will be learn and drop the packet	`bcmRxTrapL2Learn2`
No flags	By default action will be do not learn and drop the packet	`bcmRxTrapL2Learn3`

   To modify the trap actions, use `bcm_rx_trap_set` with the relevant trap code.

NOTE: API also enable/disable learning per port, see Section 21.12, MACT Management: L2 Learn.

8. To set flooding group offset call `bcm_port_flood_group_set(unit, port, flags, flood_groups)`.

   Flags are not used currently. For `flood_groups`, it is defined as below:

```
typedef struct bcm_port_flood_group_s {
    bcm_gport_t unknown_unicast_group;
    bcm_gport_t unknown_multicast_group;
    bcm_gport_t broadcast_group;
} bcm_port_flood_group_t;
```

   Members in the structure are used as offsets to flooding groups for unknown unicast, unknown multicast and broadcast packets. The value used for them is gport that can be:

   – `BCM_GPORT_BLACK_HOLE` – Drop packets on a specific InLIF or in-port. See Chapter 2, Ports and Generalized Ports, for additional configuration required for black hole.
   – `BCM_GPORT_TYPE_MCAST` – The Multicast-ID is the offset to flooding groups for Unknown packets.
   – `BCM_GPORT_TYPE_TRAP` – Flooding groups for unknown packets are destined to the configured trap (supported in the BCM88830 only).
   – `BCM_GPORT_INVALID` – Keep the flooding group unchanged.

9. To set same-interface filtering, call `bcm_port_control_set(unit, port, type,value)`.
   – `port` – Local-port
   – `type` – `bcmPortControlBridge`
   – `value` – Use 0 for disabling and 1 for enabling

10. To enable ingress/egress filtering on a port, call bcm_port_vlan_member_set(unit, port, flags)
   – `port` – Local-port or trunk gport
   – `flags` – Flags to indicate port filtering mode:
     - BCM_PORT_VLAN_MEMBER_INGRESS – Indicates ingress port filtering
     - BCM_PORT_VLAN_MEMBER_EGRESS – Indicates egress port filtering

NOTE: Both flags can be set at the same time.

21.4.3 Shell Commands

None

21.4.4 Application Reference

Example of port bridging typical settings:

- **Type:** CINT reference
- **Path:** `$SDK/src/examples/dnx/cint_dnx_vlan_membership.c`

21.5 Port x VLAN Properties

Following Port x VLAN properties are possible:

- Set VLAN is member of the port (both incoming and outgoing port).

NOTE: Unlike previous devices, VLAN APIs do not update Multicast groups and their members. This is done solely by multicast APIs.

NOTE: If egress ports are added asymmetrically to a certain VLAN, calling `bcm_vlan_destroy` does not clear them from the membership or reset the tx-tag configuration.

21.5.1 SOC Properties

None

21.5.2 Configuration Flow

To define the VLAN members ports and its TX-tag VID functionality, call:

- Option1 call:

  `bcm_vlan_port_add(unit, vlan, pbmp, ubmp);`

  `vlan` – Packet default VLAN.

  `pbmp` – Set the port bitmap that allow the VLAN to be member.

  `ubmp` – Set the port bitmap that will modify packet-header to be untagged (if the outgoing packet has a VLAN-tag that match the packet default VLAN)

NOTE: It is assumed symmetric configuration between incoming and outgoing port VLAN-membership.

- Option2 call:

  `bcm_vlan_gport_add(unit, vlan, outgoing-port, flags)`

  `vlan` – Packet default VLAN

  `outgoing-port` – Outgoing logical port

  The following `flags` operations are possible:

Flag	Description
`BCM_VLAN_GPORT_ADD_INGRESS_ONLY`	Ingress only operation
`BCM_VLAN_GPORT_ADD_EGRESS_ONLY`	Egress only operation
`BCM_VLAN_GPORT_ADD_MEMBER_REPLACE`	Change the API attribute to the opposite value

Flag	Description
BCM_VLAN_GPORT_ADD_MEMBER_DO_NOT_UPDATE	Do not update VLAN membership.
BCM_VLAN_GPORT_ADD_TAG_DO_NOT_UPDATE	Do not update TX tag functionality (only egress functionality)
BCM_VLAN_PORT_UNTAGGED	Indicates if the port should be added to untagged pbmp

21.5.3 Shell Commands

Use the `vlan show [vlan=<vlan id>]` command to show VLAN information from the BCM shell.

21.5.4 Application Reference

Example of VLAN membership typical settings

- **Type:** CINT reference
- **Path**: `$SDK/src/examples/dnx/cint_dnx_vlan_membership.c`

21.6 Port x VLAN-Tag-Structure Configuration

For the purpose of VLAN editing and VLAN classification, the user can classify the incoming packet to different formats, named TAG-format. This process is done according to VLAN-Tag-Structure. VLAN-Tag-Structure includes the TPID values on the packet's VLAN-tags and for existing priority tags (tag with VID 0), the resolution of Tag-Format is done. This process is also called LLVP. For the Ethernet bridge application, the formats are per incoming/outgoing port. That is, for each port and the parser VLAN inputs, a tag-format is picked. Parser inputs are defined according to the TPID global configuration.

The Tag-Format characterizes the Tag structure. For example, the required processing for a packet that has an untagged packet, a single S-Tag, single C-Tag, or double-tag, and so on. The Tag-Format contains fixed defined values, and the user can assign Port x TPID values to fixed Tag-Formats options. Later, Tag-format is used as an input to VLAN-editing to generate a VLAN Edit command (together with the LIF information) and for VLAN-classification to generate the right ISEM/ESEM-key.

The following configuration is possible per Port x VLAN-Tag-Structure:

- Tag-format (one of the available options)
- Drop/Admit decision

Tag-Format is an object in the SDK. Prior to use, it must be created, and its tag-format properties must be set. One of the properties is the tag structure type that defines later the match criteria to used for In-AC (VLAN-Port) and its VLAN translation.

By default, when port is added to the device, it is initialized to a profile that assumes all packets are classified as tag-format untagged and thus one profile is taken for this initialization.

NOTE: The SDK uses one combination of Ingress/Egress LLVP profile for Native LLVP processing (inner ETH VLAN editing). A special flag indicates the Native LLVP, which is beyond the scope of Ethernet bridging.

21.6.1 SOC Properties

`l2_port_tpid_class_mode` indicates the key type to the ingress LLVP table.

NOTE: The SOC property is supported in BCM88800 and BCM88480 devices.

21.6.2 Configuration Flow

1. Complete the TPID configuration, see Section 21.3, TPID Definitions. Only after TPID configuration is done is it possible to configure the VLAN tag-structure.
2. Create a VLAN tag classification ID using a call to `bcm_port_tpid_class_create(unit, flags, tag_struct_type, tag_format_class_id)`.

   This function allocates a TAG-Struct based on the TAG-Struct type and maps this TAG-Struct to proper hardware TAG-Struct value. This allocated TAG-Struct can later be used at the BCM API `bcm_port_tpid_class_set/get`

   – Flags:

Flag Name	Explanation
`BCM_PORT_TPID_CLASS_INGRESS_ONLY`	Create ingress only VLAN tag classification. If both ingress and egress unset then API will create object for both sides.
`BCM_PORT_TPID_CLASS_EGRESS_ONLY`	Create egress only VLAN tag classification. If both ingress and egress unset then API will create object for both sides.
`BCM_PORT_TPID_CLASS_WITH_ID`	Create VLAN tag classification as specified by the user.

   – `tag_struct_type`: The VLAN tag classification type. The options are:

Tag Structure Type	Explanation
`bcmTagStructTypeUntag`	Create VLAN tag classification for untagged packet
`bcmTagStructTypeSTag`	Create VLAN tag classification for single Service tag packet
`bcmTagStructTypeCTag`	Create VLAN tag classification for single Customer tag packet
`bcmTagStructTypeSCTag`	create VLAN tag classification for double tag packet (Service and customer tags).
`bcmTagStructTypeSPrioCTag`	create VLAN tag classification for double tag packet with priority (Service tag is priority and customer tags).
`bcmTagStructTypeSSTag`	Create VLAN tag classification for double tag packet (both outer and inner tags are Service tags)
`bcmTagStructTypeCCTag`	Create VLAN tag classification for double tag packet (both outer and inner tags are Customer tags)
`bcmTagStructTypeSPrioTag`	Create VLAN tag classification for a single service tag packet with priority

   – `tag_format_class_id`:pointer to the created VLAN tag classification. This field is input when the flag `BCM_PORT_TPID_CLASS_WITH_ID` is used.
3. Assign Incoming-port and Packet TPID values to Tag-format (and other LLVP values).
   Call `bcm_port_tpid_class_set(unit, tpid_class)`.
   The field `tpid_class` is a structure `bcm_port_tpid_class_t` that contains the following fields:

   Control API:

   – `tpid_class.flags`:

Tag Structure Type	Explanation
`BCM_PORT_TPID_CLASS_INGRESS_ONLY`	Only configure the ingress
`BCM_PORT_TPID_CLASS_EGRESS_ONLY`	Only configure the egress
`BCM_PORT_TPID_CLASS_NATIVE_EVE`	Native egress VLAN classification
`BCM_PORT_TPID_CLASS_NATIVE_IVE`	Native ingress VLAN classification

NOTE: When the SOC property `l2_port_tpid_class_mode` is set to 1, the API cannot be called when both ingress and egress sides are set.

Key fields:

- `tpid_class.port` – Local port
- `tpid_class.tpid1` – The outer tag TPID value, can be one of the following:
  - `BCM_PORT_TPID_CLASS_TPID_ANY` – Any TPID
  - `BCM_PORT_TPID_CLASS_TPID_INVALID` – Unidentified TPID (that is, untag)
  - One of the configured Global TPID values
- `tpid_class.tpid2` – The inner tag TPID value, can be one of the following:
  - `BCM_PORT_TPID_CLASS_TPID_ANY` – Any TPID
  - `BCM_PORT_TPID_CLASS_TPID_INVALID` – Unidentified TPID (that is, untag)
  - One of the configured Global TPID values
- `tpid_class.flags - BCM_PORT_TPID_CLASS_OUTER_IS_PRIO` – The outer identified TPID is the priority. In other words, VLAN ID equals zero. This field cannot be used when SOC property `l2_port_tpid_class_mode` is set to 1.
- `tpid_class.flags - BCM_PORT_TPID_CLASS_OUTER_NOT_PRIO` – The outer identified TPID is *not* the priority. In other words ,the VLAN ID is not zero. This field cannot be used when SOC property `l2_port_tpid_class_mode` is set to 1.

Action fields:

- `tpid_class.tag_format_class_id` – The tag-format to generate on such packet.
- `tpid_class.flags - BCM_PORT_TPID_CLASS_DISCARD` – If set, discard packet.

NOTE: On BCM init, all ports are set to the default LLVP profile that treats all packets as untagged packets.

21.6.3 Shell Commands

None

21.6.4 Application Reference

See the following functions in the `src/appl/reference/dnx/appl_ref_vlan_init.c` file:

- TPID initialization function, `appl_dnx_global_tpid_init()`

NOTE: Global TPIDs must be set prior to setting LLVP profile.

- LLVP initialization function, `appl_dnx_vlan_llvp_init()`

At application initialization and after global TPID values are set, the default LLVP profile is set for the following packets:

- Untag.
- One tag – C-tag.
- One tag – S-tag.
- Double tag – S-tag plus C-tag.
- Double tag with priority – S-tag with priority (VID = 0) plus C-tag.
- Double tag – S-tag plus S-tag.
- Double tag – C-tag plus C-tag.

[IN] **Outer TPID Value**	**[IN]** **Inner TPID Value**	**[IN]** **Outer Is Priority**	**[OUT]** **LLVP Incoming Tag-Structure**
BCM_PORT_TPID_CLASS_TPID_INVALID	BCM_PORT_TPID_CLASS_TPID_INVALID	No	DNX_PP_ETHERNET_FRAME_VLAN_FORMAT_NONE

[IN] Outer TPID Value	[IN] Inner TPID Value	[IN] Outer Is Priority	[OUT] LLVP Incoming Tag-Structure
DNX_TPID_VALUE_C_8100_TAG	BCM_PORT_TPID_CLASS_TPID_INVALID	No	DNX_PP_ETHERNET_FRAME_VLAN_FORMAT_C_TAG
DNX_TPID_VALUE_S_9100_TAG	BCM_PORT_TPID_CLASS_TPID_INVALID	No	DNX_PP_ETHERNET_FRAME_VLAN_FORMAT_S_TAG
DNX_TPID_VALUE_S_88A8_TAG	BCM_PORT_TPID_CLASS_TPID_INVALID	No	DNX_PP_ETHERNET_FRAME_VLAN_FORMAT_S_TAG
DNX_TPID_VALUE_S_9100_TAG	DNX_TPID_VALUE_C_8100_TAG	No	DNX_PP_ETHERNET_FRAME_VLAN_FORMAT_S_C_TAG
DNX_TPID_VALUE_S_88A8_TAG	DNX_TPID_VALUE_C_8100_TAG	No	DNX_PP_ETHERNET_FRAME_VLAN_FORMAT_S_C_TAG
DNX_TPID_VALUE_S_9100_TAG	DNX_TPID_VALUE_C_8100_TAG	Yes	DNX_PP_ETHERNET_FRAME_VLAN_FORMAT_S_PRIORITY_C_TAG
DNX_TPID_VALUE_S_88A8_TAG	DNX_TPID_VALUE_C_8100_TAG	Yes	DNX_PP_ETHERNET_FRAME_VLAN_FORMAT_S_PRIORITY_C_TAG
DNX_TPID_VALUE_S_9100_TAG	DNX_TPID_VALUE_S_9100_TAG	No	DNX_PP_ETHERNET_FRAME_VLAN_FORMAT_S_S_TAG
DNX_TPID_VALUE_S_88A8_TAG	DNX_TPID_VALUE_S_88A8_TAG	No	DNX_PP_ETHERNET_FRAME_VLAN_FORMAT_S_S_TAG
DNX_TPID_VALUE_C_8100_TAG	DNX_TPID_VALUE_C_8100_TAG	No	DNX_PP_ETHERNET_FRAME_VLAN_FORMAT_C_C_TAG

21.7 VLAN-Translation AC-LIF

VLAN-Port logical port is an ingress/egress L2 object, which is referred as AC (Attachment Circuit) LIF (Logical Interface). It may have different purposes:

- AC LIF may represent an AC *service* that is representing a flow of packets performing Ethernet bridging: forwarding in the same service to derive the VSI, learning is done on these packet flows to return to this service (thus the AC LIF is symmetric and the OutLIF pointed from MACT and resides at the EEDB), flooding is used when there is no knowledge.
- AC LIF may represent an AC *VLAN translation* that performs VLAN manipulations based on the incoming or outgoing interface (where an *interface* is a port or port x VLAN object). For that purpose, the AC can be asymmetric (ingress only or egress only, and the OutLIF resides at the ESEM instead of directly pointed to EEDB). Additionally, learning might be performed on the physical port level only, and flooding is used to replicate on the port-level only (i.e. one copy per physical port, copy is set to encap_id-CUD invalid).

  In addition, to reduce the number of *VLAN Translation* AC LIFs to be allocated, an optimized approach, called *optimized* VLAN translation AC LIF, is available. In this AC LIF, the VSI (service) assignment mode is set to base + VLAN, which means that the configured VSI is used as a base value (value must be 4K modulo), and the packet VLAN is added to it which leads to up to 4K different VSIs per one AC LIF. In that case multiple VLANs on the same port are reduced into one object.

This section describes how to create and configure VLAN Translation AC LIF including both the non-Optimized and Optimized modes. For a description of Service AC-LIFs, see Section 27.4, Creating Service AC-LIFs

Creating VLAN Translation AC LIF object is done by `bcm_vlan_port_create` API which returns object-ID as VLAN-Port gport. The proper AC LIF properties are initialized.

For Ingress VLAN Translation AC LIF object, a LIF is allocated in the LIF-table and based on the user parameters, proper ISEM entry is set or the default LIF for the port is set to point to that AC LIF.

For Egress, the AC data is located directly in the ESEM or at the port default-AC database, thus no EEDB LIF allocation is required. Since LIF data is not pointed but directly derived from the lookup table, the returned object-ID is a SW-ID handle, VIRTUAL object. Note that a Virtual object cannot be shared by multiple ESEM entries (because AC data is located directly in the ESEM payload).

For out-port default-AC, outgoing port is set to Out-AC default entry. This is done by having an intermediate object, Out-AC Default profile. Out-AC default profiles are exposed to the user via the API. Each port at creation is associated with a reserved Out-AC default profiles that cannot be deleted.

At Ingress, additional ISEM lookups can be added to point to this AC LIF by using the `bcm_port_match_add` API. At Egress, for port-default AC, multiple outgoing ports can be linked to the same Out-AC Default profile also by using `bcm_port_match_add` API.

On AC LIF is creation, an object-ID, named VLAN-Port gport is encoded and returned to the user to be used as a handle to the allocated AC. The encoding of this gport is software based and used by the SDK to identify the AC LIF. For detailed description of gport encoding, gport type and subtype, etc, see Section 3.2, Gport Encoding of a Logical Interface.

Per Ingress VLAN translation AC LIF, the following settings are possible to configure:

- Base-VSI and VSI assignment mode – Decide VSI in both non-optimized or optimized mode. If the mode is non-optimized, derive VSI-ID directly from the AC LIF base-VSI. For the case of optimized VLAN translation, the VSI ID is equal to base-VSI plus Packet VLAN.
- Global InLIF – Used as an object ID for software purposes only (handle for BCM APIs). Object ID is allocated from the Global InLIF range. For Global-LIF range information, see Section 7.3, Global LIF-ID Management.

NOTE: Global InLIF is only a software object in this case. It does not populate the value to the LIF table, and the object cannot be used by other stages in the pipeline.

- VLAN translation information – For details, refer to Chapter 16, VLAN Editing Mechanism.

NOTE: By default, upon Ingress VLAN translation AC LIF creation, VLAN translation default properties are set to zero (VLAN Edit Profile, VID1, VID2, QoS PCP-DEI profile)

- QoS-Profile of PHB and Remark – Indicate the QoS profile that is used to define the PHB and Remark functionality at the Ingress. For more information, see Chapter 10, QoS

NOTE: By default, upon creation of AC-LIF, Ingress QoS profile is 0. For more information, see Chapter 10, QoS.

- QoS-Model of PHB, Remark, and TTL – Indicate the type of QoS inheritance. For more information, see Chapter 10, QoS.
- Learn: enable and information – Set the destination for learning purposes. For AC LIF VLAN translation, the only possible learning is physical port (for example, local-port, system-port, flow-id, and so on). It is possible to enable/disable learning.

NOTE: By default, upon creation of VLAN-translation AC-LIF, learn-enable is true.

- Stat-PP-command and Stat-PP-profile – Assign counter pointer with stat-profile to the Ingress AC object. For more information, see Chapter 15, PP Statistics Generation.
- Unknown-DA: Possible to change offsets to flooding groups based on Unknown-Unicast-DA, Unknown-Multicast-DA, Broadcast-DA per AC-LIF.

NOTE: Only four combinations are possible for all AC LIF. By default upon AC-LIF creation profile 0 used which means offset is 0 and flooding group ID is equal to VSI (Service).

- Wide-data – General data for ACL purposes (for details, see Section 21.17, In-AC Wide Data).
- Action-profile – Determines the trap forwarding/mirror action. For more information, see Section 12.5.6, LIF Traps. By default, the action profile is None.

The following settings are possible to configure per Egress VLAN translation AC LIF:

- Virtual object ID – Used as an object ID for software purposes only (handle for BCM APIs) if the ESEM database and its types are used.
- VLAN translation information — For details, see Chapter 16, VLAN Editing Mechanism.

NOTE: By default, upon creation of AC LIF creation, VLAN translation default properties are zeros (VLAN translation action ID, VID1, VID2, QoS PCP-DEI profile).

- Action-profile – Possible to overwrite the ETPP forwarding/mirror action. For more information, see Section 12.5.6, LIF Traps.
- Stat-PP-command and Stat-PP-profile – It is possible to assign counter pointer with stat-profile to the egress AC object. For more information, see Chapter 15, PP Statistics Generation.
- Remark profile – Indicate the QoS profile that is used to define the remark functionality at the egress. For more information, see Chapter 10, QoS.

NOTE: By default, upon creation of AC-LIF, Egress QoS profile is 0.

- QOS-Model – Indicate the type of QOS Marking (Pipe, Uniform) on the Ethernet header. For more information, see Chapter 10, QoS.
- QOS-Parameters – If the QOS-model is Pipe, the PCP and DEI parameters are available to set in the VLAN-tag (according to Egress VLAN editing decision).

Replace Functionality

After an ingress or egress VLAN translation AC LIF is created, some of its settings may be modified. This operation is known as a *replace*. To perform a replace operation on an AC, the user must set all the unchanged VLAN-Port fields to their previous values and set new values for the fields to replace.

A suggested practice is to call `bcm_vlan_port_find()` to retrieve the previous configuration structure, set the desired fields, add the replace flags `BCM_VLAN_PORT_REPLACE` and `BCM_VLAN_PORT_WITH_ID` to the flags field, and call `bcm_vlan_port_create()` with the modified structure.

NOTE: A replace operation does not allow all fields to be modified. Only specific fields can be modified, depending on their functionality. For example, modifying an ISEM/ESEM lookup or a VSI Assignment mode is not allowed. In the configuration flow, assume each field in the API is not replaceable unless stated otherwise.

21.7.1 SOC Properties

None

21.7.2 Configuration Flow

21.7.2.1 Ingress AC-LIF VLAN Translation

1. Create an ingress AC LIF for VLAN Translation (optimized and non-optimized) by calling:
   `bcm_vlan_port_create(unit, vlan_port)`
   – `vlan_port.flags`:

Supported Flags	Description
`BCM_VLAN_PORT_CREATE_INGRESS_ONLY`	Create ingress only object
`BCM_VLAN_PORT_WITH_ID`	When set, provides the required allocated In-AC gport in `vlan_port_id`.
`BCM_VLAN_PORT_VSI_BASE_VID`	When set with BCM_VLAN_PORT_CREATE_INGRESS_ONLY, the API creates an “Optimized VLAN translation” object. If it is not set, a non-optimized object is created. **NOTE:** Flush group cannot be used in this case.
`BCM_VLAN_PORT_INGRESS_WIDE`	The In-AC contains general (wide) data, which can be used by the ACL. For more information refer to Section 21.17, In-AC Wide Data. This field is replaceable.
`BCM_VLAN_PORT_REPLACE`	Use in case of replace.
`BCM_VLAN_PORT_STAT_INGRESS_ENABLE`	Indicates statistics enabled on the ingress side. For more information regarding the creation of statistics interfaces and counting, refer to Chapter 15, PP Statistics Generation. This field is replaceable.

   – `vlan_port.criteria` – The criteria represents how the LIF is pointed or hit

Criteria	Match Ethertype	Lookup Information
`BCM_VLAN_PORT_MATCH_NONE`	Invalid	No match is set for the LIF. Can be later added using `bcm_port_match_add`. Can be used by the Optimized mode.

Criteria	Match Ethertype	Lookup Information
BCM_VLAN_PORT_MATCH_PORT	Invalid	Set the Incoming-port.default-LIF to this LIF. It can be used by the Optimized mode.
BCM_VLAN_PORT_MATCH_PORT	Valid	TCAM: EtherType × VLAN-Domain. It can be used by the Optimized mode.
BCM_VLAN_PORT_MATCH_PORT_UNTAGGED	Invalid	ISEM: Incoming-Port In order for a packet to match this database, the tag format of untagged needs to be hit.
BCM_VLAN_PORT_MATCH_PORT_UNTAGGED	Valid	TCAM: EtherType × VLAN-Domain In order for a packet to match this, the tag format of untagged needs to be hit.
BCM_VLAN_PORT_MATCH_PORT_VLAN	Invalid	ISEM: S-VLAN × VLAN-Domain In order for a packet to match this database, the tag format S-tag needs to be hit.
BCM_VLAN_PORT_MATCH_PORT_VLAN	Valid	TCAM: S-VLAN × EtherType × VLAN-Domain In order for a packet to match this, the tag format S-tag needs to be hit.
BCM_VLAN_PORT_MATCH_PORT_VLAN_STACKED	Invalid	ISEM: lookup of S-VLAN × C-VLAN × VLAN-Domain In order for packet to match this database, tag formats of double-tag must be hit.
BCM_VLAN_PORT_MATCH_PORT_VLAN_STACKED	Valid	TCAM: S-VLAN × C-VLAN × EtherType x VLAN-Domain In order for a packet to match this, the tag format Double-tag must be hit.
BCM_VLAN_PORT_MATCH_PORT_CVLAN	Invalid	ISEM: C-VLAN × VLAN-Domain In order for a packet to match this database, the tag format C-tag must be hit.
BCM_VLAN_PORT_MATCH_PORT_CVLAN	Valid	TCAM: C-VLAN × EtherType × VLAN-Domain. In order for a packet to match this, the tag format C-tag must be hit.
BCM_VLAN_PORT_MATCH_PORT_PCP_VLAN	Invalid	TCAM: S-VLAN × PCP × VLAN-Domain. In order for a packet to match this, the tag format S-tag or Double-tag must be hit.
BCM_VLAN_PORT_MATCH_PORT_PCP_VLAN	Valid	TCAM: S-VLAN × PCP × VLAN-Domain × Ether-type. In order for a packet to match this, the tag format S-tag or Double-tag must be hit.
BCM_VLAN_PORT_MATCH_PORT_PCP_VLAN_STACKED	Invalid	TCAM: S-VLAN × C-VLAN x PCP × VLAN-Domain. In order for a packet to match this, the tag format Double-tag need to be hit.
BCM_VLAN_PORT_MATCH_PORT_PCP_VLAN_STACKED	Valid	TCAM: S-VLAN x C-VLAN x PCP x VLAN-Domain x Ether-type. In order for a packet to match this, the tag format Double-tag must be hit.

- `vlan_port.vsi` – The Virtual Switch instance (VSI) ID (Base-VSI) which this AC LIF is associated with.
  - Must be different than zero for the non-optimized VLAN translation mode.
  - Must be modulo 4K for optimized VLAN translation mode.
  - This field is replaceable as long as its mode is not changed.

- `vlan_port.match_vlan` – Relevant for any criteria that integrates a VLAN, such as VLAN x PORT. Represents the Outer VLAN ID for the TCAM lookup.
- `vlan_port.match_inner_vlan` – Relevant only for the criteria VLAN x VLAN x PORT. Represents the Inner VLAN ID for the TCAM lookup.
- `vlan_port.port` – The physical port associated with the LIF. Used to derive the VLAN-domain and the learning information.

  Even if the match criteria is NONE, the field is used for learning information.

  This field is replaceable only for learning purposes. Therefore, a port can be changed as long as the VLAN domain does not change
- `vlan_port.group` – Flush group used by L2 flush. The field is replaceable only if it was already set at creation. The maximum value is 2047.
- `vlan_port.encap_id` – The returned LIF-ID, only for get purposes.
- `vlan_port.vlan_port_id` – The Global InLIF VLAN-Port gport (can be in/inout according to flag `BCM_VLAN_PORT_WITH_ID`)

2. Enable or disable learning by using `bcm_port_learn_set`:
   - `port` – VLAN-Port ID (Ingress AC LIF VLAN translation object created from VLAN port API)
   - `flags` – `BCM_PORT_LEARN_ARL` to enable learning otherwise disable.
   - The learning object in this case is destination only (usually system-port), without LIF.
3. Set flooding groups offset call `bcm_port_flood_group_set(unit, port, flags, flood_groups)`.
   - `port` – VLAN-Port ID (Ingress AC LIF VLAN translation object created from VLAN port API)
   - `flags` – Not used currently.
   - `flood_groups` – It is defined as follows:

     ```
     typedef struct bcm_port_flood_group_s {
        bcm_gport_t unknown_unicast_group;
        bcm_gport_t unknown_multicast_group;
        bcm_gport_t broadcast_group;
     } bcm_port_flood_group_t;
     ```

     Members in the structure are used as offset to flooding groups for unknown unicast, unknown multicast and broadcast packets. The value used for them is gport that can be:
     - `BLACK_HOLE` – Drop packets on specific InLIF or in-port. See Chapter 2, Ports and Generalized Ports, for additional configuration required for black hole.
     - `MCAST` – The Multicast-ID is the offset to flooding groups for Unknown packets.
     - `GPORT_INVALID` – Keep the flooding group unchanged.
     - `FORWARD_PORT` – Forward type of port is the destination.
     - `TRAP` – Trap type of gport is the destination (available in BCM88830).
4. To configure statistic command properties per AC object use: `bcm_gport_stat_set` with `gport=vlan` port gport as described in Chapter 15, PP Statistics Generation.
5. Configure WIde-data. For more information, see Section 21.17, In-AC Wide Data.
6. Configure Action-profile for object by calling: `bcm_rx_trap_lif_set`. See Section 12.5.6, LIF Traps.
7. Configure a QoS Map profile for the object by calling `bcm_qos_port_map_set(unit, vlan_port.vlan_port_id, qos_map_id, -1)`. For more information, see Chapter 10, QoS.

21.7.2.2 Egress AC-LIF VLAN Translation

1. Create **Egress** AC LIF for **VLAN Translation** by calling: `bcm_vlan_port_create(unit, vlan_port)`:
   – `vlan_port.flags` – BCM flags for controlling API behavior:

Flag	Description
`BCM_VLAN_PORT_CREATE_EGRESS_ONLY`	Create an egress only object
`BCM_VLAN_PORT_VLAN_TRANSLATION`	Create an egress only VLAN Translation LIF
`BCM_VLAN_PORT_DEFAULT`	Indicate the creation of port-default (using a profile). For creating ESEM-AC (non-port-default), this flag should not be set (refer to the criteria).
`BCM_VLAN_PORT_REPLACE`	Use in case of replace
`BCM_VLAN_PORT_STAT_EGRESS_ENABLE`	Enable counting on the AC object
`BCM_VLAN_PORT_VLAN_TRANSLATION_TWO_VLAN_TAGS`	Creates an egress VLAN translation LIF with up to two VLAN tags. It must be set together with the `BCM_VLAN_PORT_VLAN_TRANSLATION` flag. If the flag is unset, only up to one VLAN edit is possible.

   – `vlan_port.criteria`

Flag	Description
`BCM_VLAN_PORT_MATCH_PORT_VLAN`	Create at the ESEM with a lookup key that is VLAN-Domain - property of the port x VSI (here called VLAN)
`BCM_VLAN_PORT_MATCH_NONE`	Create at the port-default-AC (using a profile)

   – `vlan_port.vsi` – Relevant only for ESEM-AC case (VSI is used as a key to the ESEM lookup when match criteria is `BCM_VLAN_PORT_MATCH_PORT_VLAN` – VLAN is the VSI). Must be different than zero.
   – `vlan_port.port` – Relevant only for ESEM-AC case (Port is used to derive the VLAN-Domain as a key to the ESEM lookup when match criteria is `BCM_VLAN_PORT_MATCH_PORT_VLAN`)
   – `vlan_port.egress_qos_model` – Set Egress QOS model (Pipe, Uniform). For more information, see Chapter 10, QoS. This field is replaceable.
   – `vlan_port.pkt_pri` – Set PCP. This field can be used when the egress QOS model is Pipe and according to egress VLAN editing decision. This field is replaceable.
   – `vlan_port.pkt_cfi` – Set DEI. This field can be used when the egress QOS model is Pipe and according to the egress VLAN editing decision. This field is replaceable.
   – `vlan_port.vlan_port_id` – In/Out - the virtual object ID.
2. To configure statistic command properties per AC object use: `bcm_gport_stat_set` with `gport`=vlan port gport as described in Chapter 15, PP Statistics Generation.
3. Configure a QoS Map profile for the object by calling `bcm_qos_port_map_set(unit, vlan_port.vlan_port_id,-1, qos_map_id)`. For more information, see Chapter 10, QoS.

21.7.2.3 Additional Match Criteria

To add a match to an existing LIF, call the following API:

`bcm_port_match_add(unit, port, match_info)`

- `port` – Physical port, virtual port or LIF-ID in gport format.
- `match_info` – The matching information:
  - `match_info.flags` – BCM flags for controlling API behavior;

Flag	Description
BCM_PORT_MATCH_INGRESS_ONLY	Ingress match settings
BCM_PORT_MATCH_EGRESS_ONLY	Egress match settings
BCM_PORT_MATCH_NATIVE	Native (inner) ETH settings

  - `match_info.match_vlan` – Other VLAN ID to match.
  - `match_info.match_inner_vlan` – Inner VLAN ID match.
  - `match_info.port` – Match port.
  - `match_info.match` – Match criteria. The following matching criteria are available:
    - For ingress:

Criteria	Match Ethertype	Lookup Information
BCM_PORT_MATCH_PORT	Invalid	Set the Incoming-port.default-LIF to this LIF. It can be used by the Optimized mode.
BCM_PORT_MATCH_PORT	Valid	TCAM: EtherType x VLAN-Domain. It can be used by the Optimized mode.
BCM_PORT_MATCH_PORT_UNTAGGED	Invalid	ISEM: Incoming-Port For a packet to match this database, the tag format of untagged needs to be hit.
BCM_PORT_MATCH_PORT_UNTAGGED	Valid	TCAM: EtherType x VLAN-Domain For a packet to match this, the tag format of untagged needs to be hit.
BCM_PORT_MATCH_PORT_VLAN	Invalid	ISEM: S-VLAN x VLAN-Domain For a packet to match this database, the tag format S-tag needs to be hit.
BCM_PORT_MATCH_PORT_VLAN	Valid	TCAM: S-VLAN x EtherType x VLAN-Domain For a packet to match this, the tag format S-tag needs to be hit.
BCM_PORT_MATCH_PORT_VLAN_STACKED	Invalid	ISEM: lookup of S-VLAN x C-VLAN x VLAN-Domain For a packet to match this database, the tag format double-tag needs to be hit.
BCM_PORT_MATCH_PORT_VLAN_STACKED	Valid	TCAM: S-VLAN x C-VLAN x EtherType x VLAN-Domain For a packet to match this, the tag format Double-tag needs to be hit.
BCM_PORT_MATCH_PORT_CVLAN	Invalid	ISEM: C-VLAN x VLAN-Domain For a packet to match this database, the tag format C-tag needs to be hit.
BCM_PORT_MATCH_PORT_CVLAN	Valid	TCAM: C-VLAN x EtherType x VLAN-Domain. For a packet to match this, the tag format C-tag needs to be hit.
BCM_PORT_MATCH_PORT_PCP_VLAN	Invalid	TCAM: S-VLAN x PCP x VLAN-Domain. For a packet to match this, the tag formats S-tag or Double-tag need to be hit.

Criteria	Match Ethertype	Lookup Information
BCM_PORT_MATCH_PORT_PCP_VLAN	Valid	TCAM: S-VLAN x PCP x VLAN-Domain x Ether-type. For a packet to match this, the tag formats S-tag or Double-tag need to be hit.
BCM_PORT_MATCH_PORT_PCP_VLAN_STACKED	Invalid	TCAM: S-VLAN x C-VLAN x PCP x VLAN-Domain. For a packet to match this, the tag format Double-tag needs to be hit.
BCM_PORT_MATCH_PORT_PCP_VLAN_STACKED	Valid	TCAM: S-VLAN x C-VLAN x PCP x VLAN-Domain x Ether-type. For a packet to match this, the tag format Double-tag needs to be hit

- For egress:

Flag	Description
BCM_PORT_MATCH_PORT	Connect port to AC profile (the ESEM Default LIF)

21.7.3 Shell Commands

None

21.7.4 Application Reference

The following CINT examples are available:

- $SDK/src/examples/sand/utility/cint_sand_utils_vlan_translate.c
- $SDK/src/examples/cint_vlan_match_untagged_packet.c
- $SDK/src/examples/cint_vlan_match_double_tagged_packet.c
- $SDK/src/examples/cint_vlan_translate_tpid_from_packet.c
- $SDK/src/examples/sand/cint_l2_basic_bridge_with_vlan_editing.c
- $SDK/src/examples/sand/cint_vlan_translate_tpid_modify.c

21.8 Egress VLAN Membership Filter AC-LIF

Standard egress VLAN membership filtering occurs per port and VID. For details, see Section 21.4, Port Bridging Properties. The egress VLAN membership filter AC-LIF extends the egress VLAN membership filter rule to include the egress AC-LIF.

The egress VLAN membership filter AC-LIF can be created by bcm_vlan_port_create() with the VID before egress VLAN translation but after ingress VLAN translation. Thus, the AC-LIF can be used for egress VLAN translation, too. However, if a service AC must exist in the application, the service AC should be created with the flag BCM_VLAN_PORT_EXTENDED. In this case, contiguous AC entries in the encapsulation stack are seen, and the last entry should be used for the filter.

The egress VLAN membership filter AC is derived from egress exact-match, which is why it has more flexible filter parameters. It takes one ESEM lookup out of the total of two lookups. If an application already consumes two ESEM lookups, the feature can not be supported.

21.8.1 SOC Properties

None

21.8.2 Configuration Flow

1. Create the service AC by calling `bcm_vlan_port_create(unit, vlan_port)` as described in Chapter 27, Q-in-Q Bridging with the `BCM_VLAN_PORT_EXTENDED` flag.
2. Set the tag format for VLAN tags after ingress VLAN translation by calling `bcm_port_tpid_class_set(unit, tpid_class)` as described in Section 21.6, Port x VLAN-Tag-Structure Configuration to indicate the customer tag existence.
3. Create the VLAN membership filter AC-LIF by calling `bcm_vlan_port_create(unit, vlan_port)` with the following parameters:
   - `vlan_port.flags`:
     - `BCM_VLAN_PORT_CREATE_EGRESS_ONLY` – Indicates an egress AC.
     - `BCM_VLAN_PORT_VLAN_TRANSLATION` – Indicates this AC is derived from ESEM with the kinds of lookup keys determined by criteria.
   - `vlan_port.criteria`: The criteria represents how the LIF is pointed or hit.
     - `BCM_VLAN_PORT_MATCH_PORT_VLAN` – Indicates the lookup key is {port, VSI, CVID}.
4. Set the AC-LIF with the TRAP action of DROP by calling `bcm_rx_trap_lif_set(unit, flags, lif_config_p)`.
   - `flags` – Not used.
   - `lif_config_p`:
     - `lif_type` – bcmRxTrapLifTypeOutLi indicates to configure an outLIF.
     - `lif_gport` – The GPORT created in Step 3.
     - `action_gport` – GPORT that represents an action profile. In case of DROP, the action_gport can be set as for example:
       `BCM_GPORT_TRAP_SET(action_gport,BCM_RX_TRAP_EG_TX_TRAP_ID_DROP,13,0).`

21.8.3 Shell Commands

None

21.8.4 Application Reference

Example for VLAN membership AC-LIF:

- **Type:** CINT reference
- **Path:** `$SDK/src/examples/sand/cint_l2_basic_bridge_with_vlan_editing.c`
- **Function:** `l2_basic_vlan_membership_filter_after_ive_set()`

21.9 VSI Ethernet Bridging Properties

Virtual Switch Instance (VSI) interconnects between interfaces and corresponds to a Forwarding-ID (FID), which is used as a key to the MAC table. In regards to Ethernet bridging, VSI interconnect interfaces of {Port , VLAN} with multiple ports, multiple VLAN-tags, or both. Those interfaces are defined by In-ACs and Out-ACs.

For any VSI, it is also possible to connect other L2VPN interfaces such as EVPN, VPLS, VXLAN, and so on. For more information on VSI properties for those L2VPN applications and how to connect those interfaces to VSI, see the applications section.

A VSI can be an L3 routing interface (ETH-RIF) for My-MAC. For more information, see Section 22.6, ETH-RIF (L3 VSI Interface) and VRF Definition.

VSI is derived according to In-AC properties. For more information about how to derive the VSI from In-AC. VSI is a system-object pass in the system-headers of the device. It is expected to sync VSI values all over the system. At the egress, VSI is used as a key for the ESEM lookup (if there is no pointed Out-AC) and can be part of the VLAN translation resolution.

VSI has multiple object names in BCM SDK. It can be considered as a `VLAN`/`vid` field (`bcm_vlan_t`) for several Ethernet bridge APIs. It can be considered as a `vpn` field (`bcm_vpn_t`) object and as a vswitch name for L2VPN applications. VSI can be also being named as FID in relation to MACT forwarding table.

The following Ethernet bridge settings are possible to configure per VSI:

- Set Unknown-DA as a multicast-group or other destination, such as a system-port, trunk group, or forwarding group, and so on.
- STG – VSI STP topology ID. See more information in Section 21.20.13, STG.
- Configure OLP events distribution per VSI profile. Learn, transplant, age-out, ACK and refresh events can be distributed in a different manner per VSI. The distribution can be using DSPs to the other OLPs, DSPs to the CPU or DMA with callbacks to the host CPU. Several distribution profiles can be configured. By default, the distribution is set according to the learning mode. Learning modes that use DMA are configured to DMA distribution, other learning modes are configured to use OLPs (FIFO 0 of the OLP).
- Configure age-out time (aging cycles). By default MAC entries age out after eight meta cycles of the age machine. The time for age out on this VSI will be aging_cycles*<meta cycle>. See more information on aging in Section 21.10, MACT Management: Aging.
- Define a limit to the number of MAC entries per VSI.
- Stat-PP-command and Stat-PP-profile – Assign counter pointer with stat-profile to the VSI. For more information, see Chapter 15, PP Statistics Generation.

21.9.1 SOC Properties

None

21.9.2 Configuration Flow

1. To create a VSI object, two possible APIs are possible:
   - For VSI between 1-4K, call `bcm_vlan_create(unit, vsi)`
     Note: API supports WITH-ID option only.
   - For VSI between 1-4K and higher than 4K, call
     `bcm_vswitch_create(unit, vsi)` or call `bcm_vswitch_create_with_id(unit, vsi)` for WITH-ID option
     The API creates VSI object and set by default:
     - VSI's unknown-DA to be Multicast-group and its value is the VSI-ID.
     - Associate VSI to default STG topology ID (`bcm_stg_default_get`)
2. Additional properties can be set per VSI, by calling API:
   `bcm_vlan_control_vlan_set(unit, vsi, control)`
   - `control.unknown_unicast_group` – Set multicast group ID for unknown-DA.
   - `control.unknown_multicast_group, control.brodacast_group` – Must be set with the same value as `control.unknown_unicast_group.`
   - `control.aging_cycles` – Set the number of meta cycles before age out.
     By default it is according to system-configuration (eight meta cycles).
   - `control.forwarding_vlan` – FID. Must be equal to VSI.
   - `control.ingress_stat_id` – Object statistics ID (ingress).
   - `control.ingress_stat_pp_profile` – Statistics profile (ingress). Only the value range of 0 to 1 is supported.

- `control.egress_stat_id` – Object statistics ID (egress). Egress VSI statistics are not supported by the BCM88690.
- `control.egress_stat_pp_profile` – Statistics profile (egress). Egress VSI statistics are not supported by the BCM88690.
- `control.unknown_dest` – The default destination for unknown-DA, after `BCM_VLAN_FLAGS2_UNKNOWN_DEST` is set. The value used for `unknown_dest` is GPORT with its encoding, which can be one of the following types:
  - `BCM_GPORT_TYPE_FORWARD_PORT` – Type of forwarding port (FEC).
  - `BCM_GPORT_TYPE_UNICAST_QUEUE_GROUP` – Type of unicast flow ID.
  - `BCM_GPORT_TYPE_MCAST_QUEUE_GROUP` – Type of multicast flow ID.
  - `BCM_GPORT_TYPE_TRAP` – Type of trap. Supported on the BCM88830 only.
  - `BCM_GPORT_TYPE_SYSTEM_PORT_ID` – Type of system port.
  - `BCM_GPORT_TYPE_TRUNK` – Type of trunk group.

NOTE: `unknown_dest` and `unknown_unicast_group` are used but cannot be used together.

- `control.flags2` – The following flag is supported:

  `BCM_VLAN_FLAGS2_UNKNOWN_DEST` – Indicate whether `control.unknown_dest` is valid for a VSI default destination. If it is set, do not set `control.unknown_unicast_group`.

For more information regarding the creation statistics interfaces and counting, see Chapter 15, PP Statistics Generation

3. To configure OLP events distribution per VSI, use: `bcm_l2_addr_msg_distribute_set` to configure events to its distribution. For more information see the configuration-flow section in Section 21.12, MACT Management: L2 Learn.
4. To configure learn limit per VSI, use: `bcm_l2_learn_limit_set`. For more information see the configuration-flow section in Section 21.12, MACT Management: L2 Learn.

21.9.3 Shell Commands

None

21.9.4 Application Reference

TBD

21.10 MACT Management: Aging

L2 addresses (MAC addresses) can be removed from the MACT if they were not in use for a while. The mechanism to support this behavior is aging. Each MAC address holds an age state that is reduced after a configured time. In each age state it is possible to configure the following events: Age out, Delete and Refresh.

- Delete – When this option is set for an age state, the MAC address is automatically deleted by the hardware when it reaches this age state. Other devices in the system are not affected by this operation. The Delete event is configured to happen in age 0 (that is, after eight meta cycles) by default and can happen in a different age if `bcm_vlan_control_vlan_set` with `aging_cycles` was used.
- Age out – Signals to the CPU that an entry was aged out and should be deleted or to immediately delete it in the OLP. The decision whether to delete an entry is up to the CPU (if a CPU is used).
  By default, the age-out event is distributed to the OLPs and used as a delete command.
  Use `bcm_l2_addr_msg_distribute_set` to configure the age out to be distributed to the CPU (see the configuration flow).
- Refresh – Updates the age state of a MAC address on all the devices in the system back to the maximum. The event can be distributed to the OLPs or CPU. The default is OLP.

  The refresh mode is set using the `bcmSwitchMactAgeRefreshMode` switch option (see the configuration flow).

Age machine is a hardware that handles the age of the entries in the MACT. The age machine runs over all the entries in the MACT and reduces their age. Before it reduces the age, the age machine checks whether the current age requires raising one of the above events. (Age out, Delete and Refresh)

The time in seconds between runs of the age machine is called meta cycle. By default entries are deleted after eight meta cycles. A VSI can be configured to be deleted after a different number of meta cycles. By default the meta cycle equals 0 seconds, that is, aging is disabled.

21.10.1 SOC Properties

None

21.10.2 Configuration Flow

1. Configure age time for the system. Time between runs of the age machine (meta cycle). By default, a MAC entry is deleted after eight meta cycles by calling:
   `bcm_l2_age_timer_meta_cycle_set(unit, meta_cycle_time);`
   `8*meta_cycle_time` is the time until a MAC entry is deleted unless it was configured differently for the VSI. For example, meta cycle set to 9 seconds and a MAC entry is deleted after 72 seconds (eight meta cycles).

NOTE: A value of 0 disables aging and this is the default of the API.

2. Configure age time per VSI. This is done by calling *bcm_vlan_control_vlan_set with aging_cycles* field. For more information, see Section 21.20.7, VSI Ethernet Bridging Properties. By default, VSI acts as the system configuration.

NOTE: It is required to create a VSI before using the API.

3. Configure the age time per OutLIF by calling `bcm_l2_gport_control_info_set` with the `aging_cycles` field and gport of the LIF to configure.

NOTE: It is required to set the limit per OutLIF (SOC property: `l2_learn_limit_mode=VLAN_PORT`).

4. Set event distribution per VSI by calling `bcm_l2_addr_msg_distribute_set`. For more information, see the configuration-flow section in Section 21.12, MACT Management: L2 Learn.

5. Configure the age refresh mode to be triggered by the DA hit, SA hit, or a combination between them.
   - `bcm_switch_control_set(unit, type, arg)`
   - `type` = `bcmSwitchMactAgeRefreshMode`
   - `arg` – Argument to the switch control's operation, which is one of the flags in the following table.

Flags	Description
`bcmSwitchAgeRefreshModeSrc`	The age of the MAC entry will be refreshed only on SA lookup.
`bcmSwitchAgeRefreshModeDest`	The age of the MAC entry will be refreshed only on DA lookup.
`bcmSwitchAgeRefreshModeSrcAndDest`	The age of the MAC entry will be refreshed on SA and DA lookup.
`bcmSwitchAgeRefreshModeSrcOrDest`	The age of the MAC entry will be refreshed on SA or DA lookup.(default setup)

21.10.3 Shell Commands

l2 info: Get the aging configuration of the system.

21.10.4 Application Reference

TBD

21.11 MACT Management: Flush Machine

The flush machine is hardware that runs over the LEM and, according to filter rules, can retrieve in CPU, delete, or change entries quickly by using dedicated hardware. Delete-all, traverse and replace APIs for LEM databases use the flush machine.

Software filters can be applied on the results for additional filtering that cannot be configured in the hardware flush machine rules. Filters in the software are slower since the CPU has to receive all the entries and filter them.

The entries coming from the flush machine to the CPU are copied to the CPU's memory using DMA hardware. The DMA hardware reduces the load from the CPU.

A dedicated offload buffer for fast DMA evacuation is allocated at device init (see Section 21.11.1, SOC Properties). The use of this buffer prevents flush machine HW from being halted by the DMA, thus preventing contention between traverse functionality and the aging mechanism.

Delete and replace operations may be applied without a user callback being triggered. Non-blocking mode (that is, the API does not wait for the flush machine to finish) is also applicable.

Non-blocking mode is also supported for traverse operations. In that case, the user callback function is called from a dedicated low-priority thread. Calling the user callback function with a NULL L2 address pointer signals the end of the traverse operation.

NOTE: If the flush machine or an aging cycle are in process, any APIs that are called will fail with a busy error code.

21.11.1 SOC Properties

The `l2_flush_buffer_nof_entries` SOC property sets the host memory size for the DMA offloading buffer in the number of MACT entries.

- If the value is greater than or equal to 128, it indicates the size of the buffer (in number of entries) used for Flush DMA copied entries.
- If the value is –1, the maximum number of entries (for the used MDB profile) will be used.
- The value range is 0 to 127 or invalid.
- The default value is –1.

21.11.2 Configuration Flow

Delete/Replaces entries according to a filter rules and actions.

`bcm_l2_replace(unit, flags, match_addr, new_module, new_port, new_trunk)`

- `flags` – Rules for the replace.

Flag Indicate Rules	Description
`BCM_L2_REPLACE_MATCH_STATIC`	Replace also static entries, otherwise replace only static or dynamic entries according to `match_addr.flags` (BCM_L2_STATIC). If `match_addr` is NULL then the logic is the same as `BCM_L2_REPLACE_MATCH_STATIC`
`BCM_L2_REPLACE_MATCH_MAC`	Compare MAC address in the filter rules
`BCM_L2_REPLACE_MATCH_DEST`	Compare destination
`BCM_L2_REPLACE_MATCH_VLAN`	Compare VSI
`BCM_L2_REPLACE_MATCH_GROUP`	Compare group field
`BCM_L2_REPLACE_IGNORE_DES_HIT`	If present, ignore hit status in compare. Otherwise replace only entries with hit status that matches `match_addr.flags`
`BCM_L2_REPLACE_IGNORE_PENDING`	Non-blocking update or delete. The API does not wait for the flush machine to finish. **NOTE:** `BCM_L2_REPLACE_NO_CALLBACKS` must be also set. **NOTE:**
`BCM_L2_REPLACE_VLAN_AND_VPN_TYPE`	If present, replace payloads of IVL entries; otherwise, replace payloads of default FWD MACT entries.

- Actions (some options can be set together).

Flag Indicate Actions (Multiple Options Can Be Set Together)	Description
`BCM_L2_REPLACE_DELETE`	Delete matching entries
`BCM_L2_REPLACE_DES_HIT_CLEAR`	Clear the hit status of the matching entries
`BCM_L2_REPLACE_DISCARD_SRC_SET`	Not supported
`BCM_L2_REPLACE_DISCARD_SRC_CLEAR`	Not supported
`BCM_L2_REPLACE_DYNAMIC_SET`	Update the matching entries to be dynamic
`BCM_L2_REPLACE_DYNAMIC_CLEAR`	Update the matching entries to be static
`BCM_L2_REPLACE_MIGRATE_PORT`	Replace the port in entry
`BCM_L2_REPLACE_NEW_TRUNK`	Replace the destination with a given trunk ID
`BCM_L2_REPLACE_PROTECTION_RING`	Replace the destination. Same as `BCM_L2_REPLACE_MIGRATE_PORT`
`BCM_L2_REPLACE_NO_CALLBACKS`	Perform replace without calling the user callback for every replaced entry. Using this flag allows the changes to be done in HW only without the overhead of the SW

- match_addr – Used if BCM_L2_REPLACE_MATCH_MAC was set in the flags. match_addr is the address to compare.
- new_port – Used when BCM_L2_REPLACE_MIGRATE_PORT is set as the new system port destination.

Delete all entries with destination of a given port:

```
bcm_l2_addr_delete_by_port(unit, mod, port, flags)
```

- port – System port for match filter
- flags – Rules for deleting the entries. The following table describes the available flags.

Table 83: bcm_l2_addr_delete_by_port Flags

Flags Indicate Rules	Description
BCM_L2_DELETE_STATIC	Also delete static entries; otherwise, delete only dynamic entries.
BCM_L2_DELETE_NO_CALLBACKS	Perform the delete action without calling the user's callback for every deleted entry. Using this flag allows the changes to be done in HW only without the overhead of the SW.
BCM_L2_DELETE_IVL	Delete only IVL entries; otherwise delete only FWD MACT entries by default.

Delete all entries with a given MAC address:

```
bcm_l2_addr_delete_by_mac(unit, mac, flags)
```

- mac – Address to use in the match filter
- flags – Rules for deleting the entries. The flags are the same as for the bcm_l2_addr_delete_by_port() API (see Table 83), in which the flags indicate rules.

Delete all entries of a VSI:

```
bcm_l2_addr_delete_by_vlan(unit, vid, flags)
```

- vid – VSI to use in the traverse filter
- flags – Rules for deleting the entries. The flags are the same as for the bcm_l2_addr_delete_by_port() API (see Table 83), in which the flags indicate rules.

Delete all entries with destination of a given trunk:

```
bcm_l2_addr_delete_by_trunk(unit, tid, flags)
```

- tid – Trunk id to use in the match filter
- flags – Rules for deleting the entries. The flags are the same as for the bcm_l2_addr_delete_by_port() API (see Table 83), in which the flags indicate rules.

Delete all entries by MAC address and port (same behavior like the previous APIs):

```
bcm_l2_addr_delete_by_mac_port(unit, mac, mod, port, flags)
```

Delete all entries by VLAN and port:

```
bcm_l2_addr_delete_by_vlan_port(unit, vid, mod, port, flags)
```

Delete all entries by VLAN and trunk

```
bcm_l2_addr_delete_by_vlan_trunk(unit, vid, tid, flags)
```

Non-Blocking Mode:

Replace can be done in a non-blocking mode – bcm_l2_replace_match or bcm_l2_replace

In this case, the API will not wait for the flush machine to finish its work. Non-blocking can be applied only to replace or delete dynamic entries.

To enable non-blocking mode, set BCM_L2_REPLACE_IGNORE_PENDING in the flags parameter, along with the BCM_L2_REPLACE_NO_CALLBACKS flag. In addition, BCM_L2_REPLACE_MATCH_STATIC must be unset. Also match_addr → flags and BCM_L2_STATIC must be 0.

MACT operations cannot be applied until the non-blocking is done.

Use `bcm_switch_control_get` with `type = bcmSwitchTraverseCommitDone` to check if the flush is done.

Replace non-blocking in bulk mode is also supported.

a. Use `bcm_switch_control_set` to set the bulk mode (`bcmSwitchTableUpdateRuleAdd`).
b. Perform the replace by calling `bcm_l2_replace` or `bcm_l2_replace_match` (same as in regular mode above).
   Replace APIs can be called a few times. Up to 16 rules are in the flush DB.
c. Use `bcm_switch_control_set` to start the flush machine (`bcmSwitchTraverseMode` + `bcmSwitchTableUpdateRuleCommitNonBlocking`).
d. Use `bcm_switch_control_get` to check if the flush is *done* in a loop with an appropriate delay (`bcmSwitchTraverseCommitDone`).
e. Use `bcm_switch_control_set` to clean rules DB (`bcmSwitchTableUpdateRuleClear`).

Traverse all the MAC table and get a call to a callback on the entries.

```
bcm_l2_traverse(unit, trav_fn, user_data)
```

- `trav_fn` – Callback that is called on every entry in the MAC table.
- `user_data` – User context that is added as a parameter to the user callback's calls.

Traverse the MAC table according to filter rules and get a call to a callback on the entries.

```
bcm_l2_match_masked_traverse(unit, flags, match_addr, mask_addr, trav_fn, user_data)
```

- `flags` – Rules for the traverse filter.

Traverse Flags	Description
`BCM_L2_TRAVERSE_MATCH_STATIC`	Traverse static and dynamic entries, otherwise traverse only static or dynamic entries according to `match_addr.flags` (`BCM_L2_STATIC`)
`BCM_L2_TRAVERSE_MATCH_MAC`	Filter by a given mac address taken from `match_addr`
`BCM_L2_TRAVERSE_MATCH_VLAN`	Filter by a given VSI taken from `match_addr`
`BCM_L2_TRAVERSE_MATCH_DEST`	Filter by a given port and `encap_id` taken from `match_addr`
`BCM_L2_TRAVERSE_MATCH_GROUP`	Filter according to the group field
`BCM_L2_TRAVERSE_IGNORE_DES_HIT`	If present, ignore hit status in compare. Otherwise traverse only entries with hit status that match `match_addr.flags`
`BCM_L2_TRAVERSE_IGNORE_PENDING`	Non-blocking get.The API does not wait for the flush machine to finish. Get results that arrive on a low-priority thread. Returns only entries that are dynamic and come from the HW MACT (no shadow). The last call to the user's callback returns with a NULL in the `bcm_l2_addr_t` pointer. The NULL signals the user that the traverse is done. Until the traverse is done, MAC operations are not allowed to run in parallel.
`BCM_L2_TRAVERSE_MATCH_IVL`	If present, traverse over the IVL entries; otherwise, traverse over the default FWD MACT entries.

Traverse can be done in a non-blocking mode using `bcm_l2_matched_traverse` and `bcm_l2_traverse`. In this case, the API will not wait for the flush machine to finish its work. The results arrive on a low-priority thread, and only entries that are dynamic and come from the HW MACT (no shadow) are returned. The last call to the user's callback returns with a NULL in the `bcm_l2_addr_t` pointer. The NULL signals to the user that the traverse is complete. To enable this non-blocking mode, `BCM_L2_TRAVERSE_IGNORE_PENDING` must be set in the `flags` parameter. MACT operations cannot be applied until the non-blocking is finished.

Replace with the ability of mask functionality:
`bcm_l2_replace_match(unit, flags, match_addr, mask_addr, replace_addr, replace_mask_addr)`

- `flags` – See parameters' definitions at `bcm_l2_replace` above.
- `match_addr` – See parameters' definitions at `bcm_lv2_replace` above.
- `mask_addr` – Mask for `match_addr`. Bits that are set in this struct are compared.
- `replace_addr` – Values for the replace operation
- `replace_mask_addr` – Mask for the `replace_addr`. Bits that are set in this struct are replaced.

For flush in blocking mode, the information described previously is true for a single operation. For a bulk operation (multiple rules on same run), the following is required:

- Collect rules without applying them.
  `bcm_switch_control_set(unit, bcmSwitchTraverseMode, bcmSwitchTableUpdateRuleAdd)`
- Add replace and delete operations without callbacks
  `bcm_l2_replace` and `bcm_l2_addr_delete_by_*` as above. Traverse APIs cannot be used in bulk mode.
- Apply the given rules on the MACT
  `bcm_switch_control_set(unit, bcmSwitchTraverseMode, bcmSwitchTableUpdateRuleCommit)`
- Clear the rules
  `bcm_switch_control_set(unit, bcmSwitchTraverseMode, bcmSwitchTableUpdateRuleClear)`
- Return the flush machine to a regular mode (one rule per operation)
  `bcm_switch_control_set(unit, bcmSwitchTraverseMode, bcmSwitchTableUpdateNormal)`

21.11.3 Shell Commands

`l2 show`

Use the flush machine to retrieve all the MACT entries.

`l2 clear all`

Use the flush machine to delete all the MACT entries.

21.11.4 Application Reference

TBD

21.12 MACT Management: L2 Learn

L2 addresses (MAC addresses) can be learned on demand when new addresses are introduced to a device.

A source address (SA) of a packet coming from an incoming interface (for example, in-port, InLIF) is learned. Later when this SA is the destination address of a packet, the packet's destination is known and no flooding is needed.

For each packet that should be learned a key and an expected payload are created. The key is looked up in the MACT. If the key is found we get an accepted payload. The expected and accepted payloads are compared to check if a transplant should be made (new destination).

If the MACT's key lookup is not found, a learn event should be created. The learn info is coming from the incoming interface information.

Learn events can be distributed in one of three ways:

1. DSPs (Dune special packets) to the OLPs in the system. Learn is done in hardware by messages that are distributed to the devices in the system.
2. DSPs to the CPU. Learn messages to the CPU that should add the MAC entries at will. When using this option, access to the DSP format document is required.
3. Callbacks calls to CPU handlers. Events from the device are copied using a DMA hardware to the host's memory. When a time threshold or after number of events threshold are crossed a callback is called to handle the learn events. Note: DMA can handle a much slower rate of DSPs.

OLP is a hardware block that should be configured to the different events distributions. The type of messages coming from the OLP can be configured to be with or without Ethernet header. It will always have ITMH. Messages coming to the OLP can have OTMH and might have Ethernet headers. Any combination of the two optional headers is possible.

Per VSI, per event, the OLP can be configured to a different distribution.

A limit to the number of entries can be set for the whole MAC table, per VSI. All the configured limits are checked and if one of them is met, the new entry that crossed the limit is not added. When setting the limit per LIF, consider the following information:

- The LIF limit is per OutLIF.
- Aging per VSI is impossible in this mode. Update the meta cycle time to control the aging time.
- Flush machine updates do not update the counters. When updating the LIFs using the flush machine, the counters will show a wrong count to the number of entries per LIFs.
- Distribution can be used without VSI (that is, `distribution.vid = -1`), which will update the default profile's distribution.

By default, only bridged packets are learned. L2 learning can also be enabled on routed packets. The decision is global (see `bcmSwitchL3RoutedLearn`). When enabled, the IP-host table cannot be used for forwarding or RPF. The SDK will return an error when adding, finding, or deleting host entries using the host APIs (`bcm_l3_host_*` APIs). In this case, the host entries should be treated as LPMs and will be stored in the LPM memory.

NOTE: IPv4 UC and IPv6 UC must be enabled together when enabling route learning per protocol.

21.12.1 SOC Properties

SOC Name	Highlights
`learning_fifo_dma_buffer_size`	Size of host memory for DMA learn target in bytes. 0 disables the DMA. Supported values: Range 20 to 327,680. Default is 200,000.
`learning_fifo_dma_timeout`	Time in microseconds from event copy by DMA until an interrupt is raised. 0 means no timeout. Supported values: Range 0 to 65,535. Default is 32,767.
`learning_fifo_dma_threshold`	Number of events copied by the DMA before raising an interrupt. Interrupt is raised according to the minimum of the number of events and the timeout. Supported values: Range 1 to 16,384. Default is 4.
`ucode_port_<port number>`	Configure ports for the OLP. For details, refer to BCM88690 *Traffic Manager Programming Guide*, Port Provisioning section.
`l2_learn_limit_mode`	System mode that defines whether the limit should be per VSI or LIF. Supported values: VLAN (default) - per VSI. VLAN_PORT - per LIF.
`l2_dma_cpu_learn_thread_priority`	Priority of the L2 CPU learning dedicated thread. Supported values: Range 0 to 65,535. Default is 50.

21.12.2 Configuration Flow

On device initialization, it is expected:

1. Configure learn mode. The reference application is setting the learn mode to ingress distributed.
   `bcm_switch_control_set(unit, type, arg)`
   – `type` = `bcmSwitchL2LearnMode`.
   – `arg` – Argument to the switch control's operation. For changing the learn mode, the arg should get the learn mode.

Traverse Flags	Description
`BCM_L2_INGRESS_CENT`	Distributes the events according to ingress OLP. Learn in SW using CPU.
`BCM_L2_INGRESS_DIST`	Distribute the events according to ingress OLP. Learn in HW using OLP.
`BCM_L2_LEARN_CPU`	Should be set in addition to `BCM_L2_INGRESS_CENT` or `BCM_L2_EGRESS_CENT` for DMA learning with callbacks. DMA copies the events to the host and a user callback is called to handle the event.

NOTE: The learning profile and related configurations are reset to default when the learning mode is changed.

2. Set a limit to the number of entries in the MAC table. Use –1 for the maximum capacity (default configuration).
   `bcm_l2_learn_limit_set(unit, limit)`
   – `limit.flags` – Set the type of limit. For a limit to the whole MAC table use `BCM_L2_LEARN_LIMIT_SYSTEM`.
   – `limit.limit` – Limit value.
3. It is also possible to enable or disable inhibiting learning events when a limit is reached:
   `bcm_switch_control_set(unit, type, arg)`
   – `type` – `bcmSwitchL2LearnLimitToCpu`
   – `arg` – 1 means limits are checked. 0 means limits are not enforced for an add MACT entry.
4. It is also possible to enable or disable limit checking for a static entry by using the following API:
   `bcm_switch_control_set(unit, type, arg)`
   – `type` – `bcmSwitchL2LearnLimitCheckStatic`
   – `arg`
   - 1 – Enable limit checking for a static entry (default).
   - 0 – Disable limit checking for static entry.
5. To enable the learning of routed packets types, call:
   `bcm_switch_control_set(unit, bcmSwitchL3RoutedLearn, arg);`
   Set `arg` as a bitmap of the following flags:

Value Options	Description
`BCM_SWITCH_CONTROL_L3_LEARN_IPV4_UC`	Enable learning for IPv4 Unicast packets
`BCM_SWITCH_CONTROL_L3_LEARN_IPV4_MC`	Enable learning for IPv4 Multicast packets
`BCM_SWITCH_CONTROL_L3_LEARN_IPV6_UC`	Enable learning for IPv6 Unicast packets
`BCM_SWITCH_CONTROL_L3_LEARN_IPV6_MC`	Enable learning for IPv6 Multicast packets
`BCM_SWITCH_CONTROL_L3_LEARN_MPLS`	Enable learning for MPLS packets

6. Configure the header of the DSP messages
   `bcm_l2_learn_msgs_config_set(unit, learn_msg_config)`
   – `learn_msg_config.flags` – Instructions for the header's configuration.

Flag	Description
BCM_L2_LEARN_MSG_DEST_MULTICAST	When set the destination of the learning message is a multicast group. dest_group is used, if it is not present then the destination is uc and dest_port is used
BCM_L2_LEARN_MSG_ETH_ENCAP	Learn messages are encapsulated in the Ethernet header. Use the specified Ethernet parameters.
BCM_L2_LEARN_MSG_LEARNING	Configure messages coming from FIFO 0 of the OLP. This FIFO is used for messages to OLP ports in HW learning mode.
BCM_L2_LEARN_MSG_SHADOW	Configure messages coming from FIFO 1 of the OLP. This FIFO is used for messages to the host CPU when in centralized learning mode

  - BCM_L2_LEARN_MSG_DEST_MULTICAST – When set the destination of the learning message is multicast group. 'dest_group' is used, if it is not present then the destination is uc and 'dest_port' is used.
  - BCM_L2_LEARN_MSG_ETH_ENCAP – Learn messages are encapsulated in Ethernet header. Use the specified Ethernet parameters.
  - BCM_L2_LEARN_MSG_LEARNING – Configure messages coming from FIFO 0 of the OLP. This FIFO is used for messages to OLP ports in hardware learning mode.
  - BCM_L2_LEARN_MSG_SHADOW – Configure messages coming from FIFO 1 of the OLP. This FIFO is used for messages to the host CPU when in centralized learning mode.
- dest_port – Destination port. Used when BCM_L2_LEARN_MSG_DEST_MULTICAST is not set in the flags field.
- dest_group – Destination multicast group. The field is used when BCM_L2_LEARN_MSG_DEST_MULTICAST is set in the flags field.
- vlan – Used when Ethernet header is configured. Set to invalid for untagged Ethernet.
- tpid – VLAN TPID
- vlan_prio – VLAN priority
- ether_type – EtherType value.
- src_mac_addr – Source MAC address.
- dst_mac_addr – Destination MAC address.
- priority – Traffic class.
- color – Drop precedence.

NOTE: This API configures both the transmitted and received header format of the OLP. If this API is called more than once, the last API call will determine the expected header format for the OLP.

7. Action for weaker over stronger transplant. It is possible to define a trap action for the case that the learned entry strength is weaker than the existing entry strength. This configuration is with the following API.
   bcm_l2_learn_action_resolution_set(unit, action_res)

- action_res – Pointer to bcm_l2_learn_action_resolution_t struct:
  - gport – Gport trap action or BCM_GPORT_INVALID
  - existing_strength – Should be set to 0
  - learn_strength – Should be set to 0

Per port, it is possible to enable and disable learning on the Ethernet ports.bcm_port_learn_set(unit, port, flags)

- port – Logical port (incoming-port) to update
- flags – Set the bit BCM_PORT_LEARN_ARL to enable learning (otherwise learning will be disabled).

NOTE: API also set relevant trap action profile for MAC SA not found (bcmRxTrapL2LearnXXX). For more information, see Section 21.20.2, Port Bridging Properties. By default, all Ethernet ports enable learning.

The following actions are possible per VSI:

- Configure the VSIs' distribution.
  `bcm_l2_addr_msg_distribute_set(unit, distribution)`
  - `distribution.flags` – Configure OLP events to the distribution queues. Set the events to configure and to which queue to distribute them.

    Events:

Flag – Events	Description
`BCM_L2_ADDR_DIST_LEARN_EVENT`	Learn events
`BCM_L2_ADDR_DIST_STATION_MOVE_EVENT`	Transplant events
`BCM_L2_ADDR_DIST_AGED_OUT_EVENT`	Age out events
`BCM_L2_ADDR_DIST_REFRESH_EVENT`	Refresh events
`BCM_L2_ADDR_DIST_ACK_EVENT`	ACK events

    Distribution:

Flag – Distribution	Description
`BCM_L2_ADDR_DIST_SET_LEARN_DISTRIBUTER`	FIFO 0 (OLP ports)
`BCM_L2_ADDR_DIST_SET_SHADOW_DISTRIBUTER`	FIFO 1 (CPU)
`BCM_L2_ADDR_DIST_SET_CPU_DMA_DISTRIBUTER`	Use DMA and callbacks. Refer to the DMA learn configure below.
`BCM_L2_ADDR_DIST_SET_NO_DISTRIBUTER`	Do not distribute the events

  - `distribution.vid` – VSI to configure
- Set a limit per VSI. Configure the maximum number of MAC entries with a given VSI.
  `bcm_l2_learn_limit_set(unit, limit)`
  - `limit.flags` – Configure which type of limit is used. For limit per VSI use `BCM_L2_LEARN_LIMIT_VLAN`
  - `limit.limit` – Configured limit number of entries

Per OutLIF:

1. Set the `l2_learn_limit_mode` SOC property to `VLAN_PORT`
2. Configure the LIF distribution.
   `bcm_l2_addr_msg_distribute_set(unit, distribution)`
   - `distribution.flags` – `BCM_L2_ADDR_DIST_GPORT`. Set distribution events per LIF. Other event and distribution flags are the same as with the VSI distribution configuration.
   - `distribution.gport` – gport of the LIF to configure.
3. Set the limit per OutLIF.

   `bcm_l2_learn_limit_set(unit, limit)`
   - `limit.flags` – Configure which type of limit is used. For limit per LIF use `BCM_L2_LEARN_LIMIT_PORT`
   - `limit.limit` – Configured limit number of entries
   - `limit.port` – Gport of the LIF

DMA learning:

1. Configure DMA learning. In this mode the OLP events are copied to the host memory using DMA. A user callback is called when there are available events:
   a. First configure the SOC properties for the DMA: `learning_fifo_dma_buffer_size`, `learning_fifo_dma_timeout`, `learning_fifo_dma_threshold`
   b. Second set the learn mode (centralized + CPU): `bcm_switch_control_set` with centralized and CPU flags. See more details in the Configure Learn Mode information previously described in Step 1.
   c. Third and last register a callback to handle the learn events:
   `bcm_l2_addr_register(unit, callback, userdata)`
      - `callback` – User callback for handling the OLP events.
      - `userdata` – Parameter that will be passed to the user's callback.
2. Cancel the DMA learning configuration.
   – Set the learn mode to a non CPU mode: `bcm_switch_control_set` with centralized or distributed and without CPU flags. See more details in Configure Learn Mode information previously described in Step 1.
   – Unregister all the previously registered learn callback:
   `bcm_l2_addr_unregister(unit, callback, userdata)`
      - `callback` – User callback to unregister.
      - `userdata` – Should point to the same context given in the register call.
3. Clear the existing L2 learn callbacks, call `bcm_l2_clear(unit)`.

21.12.3 Shell Commands

Use `l2 info` to show the learn mode.

21.12.4 Application Reference

Distributed learning configuration of the OLP

Example of application configuring the header of the DSPs coming out of FIFO 0 in the OLP. FIFO 0 of the OLP is used in distribution learn mode. The packets distributed by this FIFO are destined to a multicast group including all the OLP ports in the system. This configuration must be done by the user in order to enable distributed learning.

Use bcm_l2_learn_msgs_config_set

- **Type:** Default Application Reference
- **Path**: `$SDK/src/appl/reference/dnx/appl_ref_l2_init.c`
- **Function** `appl_dnx_olp_init()`

Example of simple bridge learning configuration

- **Type:** CINT reference
- **Path**: `l2_basic_bridge_run` in `$SDK/src/examples/sand/cint_l2_basic_bridge_with_vlan_editing.c`

21.13 MACT Management: Registration upon MACT Changes

SDK user can register callbacks to be updated on different types of changes in the MAC table. Changes that can be monitored are those done by the flush machine or coming from the DMA queue of the OLP. (see Section 21.20.10, MACT Management: L2 Learn DMA learning configuration) Supported changes for monitoring are: delete, add, learn, age out, and transplant (move).

21.13.1 SOC Properties

None

21.13.2 Configuration Flow

Register a callback to handle the MAC table reports.

```
bcm_l2_addr_register(unit, callback, userdata)
```

- `callback` – User callback for handling the events.
- `userdata` – Parameter that will be passed to the user's callback.

The registered callback should have the following prototype:

```
typedef void (*bcm_l2_addr_callback_t)(
       int unit,
       bcm_l2_addr_t *l2addr,
       int operation,
       void *userdata);
```

- `l2addr` – Associated MAC entry
- `operation` – Type of event. One of the following:
  - `BCM_L2_CALLBACK_DELETE`
  - `BCM_L2_CALLBACK_ADD`
  - `BCM_L2_CALLBACK_LEARN_EVENT`
  - `BCM_L2_CALLBACK_AGE_EVENT`
  - `BCM_L2_CALLBACK_MOVE_EVENT`
- `userdata` – Pointer given in the `bcm_l2_addr_register`.

21.13.3 Shell Commands

None

21.13.4 Application Reference

TBD

21.14 L2 MACT Forwarding

The MACT entries can be added statically and dynamically. When referring to MACT, the SDK can configure either the FWD MACT table (regular MACT) or FWD MACT IVL table (dedicated table).

The following settings are possible to configure per MACT entry:

- Destination – Physical destination (for example, system-port, local-port, flow-id) or FEC
- Global OutLIF (optional) – Can be a pointer to encapsulation database for example, VLAN-Port(AC), MPLS-Port (PWE-LIF), and so on.
- Group – Flush-machine grouping parameter.
- Indicate Static or Dynamic entry

21.14.1 SOC Properties

None

21.14.2 Configuration Flow

Initialize MAC entry

`bcm_l2_addr_t_init(l2addr, mac_addr, vsi)`

- `l2addr` – Struct to init.
- `mac_addr` – MAC address for the `l2addr` param.
- `vsi` – VSI for `l2addr` param.

Add MAC entry

`bcm_l2_addr_add(unit, l2addr)`

NOTE: This API overrides the existing values if the VSI and MAC pair already exists.

- `l2addr.flags` – Configuration flags for the l2 address.

Flag	Description
`BCM_L2_STATIC`	MAC address configured to be static. Entry does not age out. If the flag is unset, the entry is added as dynamic.
`BCM_L2_SRC_HIT`	Indicates whether the MAC entry was hit by an SA lookup. This flag is set only in `bcm_l2_addr_get`. An aging meta cycle resets the source hit bit.
`BCM_L2_DES_HIT`	Indicates if the MAC entry was hit by a lookup. This flag is set only in `bcm_l2_addr_get`.
`BCM_L2_HIT`	No longer supported. This API returns a hit for SRC/DST separated by `BCM_L2_DES_HIT` and `BCM_L2_SRC_HIT`
`BCM_L2_TRUNK_MEMBER`	Indicates that the destination is a trunk. The tgid field is used to provide the `trunk-id`, or port field is used to provide the trunk gport.
`BCM_L2_MCAST`	Indicates multicast entry. In that case destination multicast is used by the `l2mc_group` field.
`BCM_L2_DISCARD_DST`	Configure the destination to be a trap from type `bcmRxTrapDfltDroppedPacke`t.
`BCM_L2_DISCARD_SRC`	Configure the destination to be a trap from one of the following types: ■ `bcmRxTrapL2DiscardMacsaFwd` ■ `bcmRxTrapL2DiscardMacsaDrop` ■ `bcmRxTrapL2DiscardMacsaTrap` ■ `bcmRxTrapL2DiscardMacsaSnoop`

Flag	Description
BCM_L2_MOVE_PORT	Used together with BCM_L2_STATIC to configure a MAC entry that does not age-out (as static entries) but can be transplanted (updated) with a new destination. A PMF program should be configured to trap the packet for a transplant (see cint_field_static_sa_move.c for an example).
BCM_L2_FLAGS2_NO_MOVE	MAC address that is *dynamic non-transplantable* and may age out. Should be set in l2addr.flags2.

- l2addr.mac – MAC address.
- l2addr.vid – VSI (FID).
- l2addr.port – Can be configured to physical port (for example, system-port, local-port, flow-id) or logical gport (for example, VLAN-Port gport).

NOTE: On bcm_l2_addr_get, if the port is a trunk, the returned value is in l2addr.port and l2addr.tgid, and the BCM_L2_TRUNK_MEMBER flag is set. If the port is mcast, then the returned value will be l2mc_group, and the BCM_L2_MCAST flag will be set.

  - l2addr.group – Group value that can be used as a flush filter parameter.
  - l2addr.modid – Used in IVL to configure the VLAN ID. Should be 0 in SVL.
  - l2addr.l2mc_group – MC group destination (if flag BCM_L2_MCAST is set)
  - l2addr.tgid – Trunk group. If BCM_L2_TRUNK_MEMBER is used, set the trunk group
  - l2addr.encap_id – Out logical interface. Can be used if port is physical-port type and a logical-interface pointer is required as well. Useful for the case of multidevices (when the logical-gport does not exist in the local device).
  - l2addr.age_state – Number of seconds until the address ages out. The age_state value is not accurate. There might be an error of ⅛ × [age-out time]. For example, if the maximum age-out time of the entry is 64 seconds, the age_state field might be wrong by ±8 seconds.
  - l2addr.learn_strength – Learn strength in l2 entry with a range of 2 to 3. When set, the entry is static.
  - l2addr.age_profile – Age profile in l2 entry that determines if the entry is subject to aging.
    - 0: Entry is not subject to aging
    - 1: Entry is subject to aging

Other utilities that are possible:

- Get an entry from the MACT (SVL only)
  bcm_l2_addr_get(unit, mac_addr, vid, l2addr)
  - mac_addr – MAC address to fetch.
  - vid – VSI to fetch.
  - l2addr – Fetched entry.
- Delete an entry from the MACT (SVL only) according to MAC address and VSI.
  bcm_l2_addr_delete(unit, mac_addr, vid)
  - mac_addr – MAC address.
  - vid – VSI.
- Get a MAC entry from FWD_MACT or FWD_MACT_IVL.
  FWD_MACT_IVL is used if modid is not zero; otherwise, FWD_MACT is picked.
  bcm_l2_addr_by_struct_get(unit, l2addr)
  - l2addr.mac – MAC address.
  - l2addr.vid – VSI (FID).
  - l2addr.modid – VLAN ID if it is not 0. When modid is zero, read the entry from FWD_MACT; otherwise, FWD_MACT_IVL.

  - `l2addr.age_state:`
    - Used as an output parameter when `BCM_L2_FLAGS2_AGE_GET` is set, indicating the estimated number of seconds before age-out.
    - In software aging mode (hardware aging disabled), used as an input parameter, indicating the software meta cycle time in seconds.
- Delete MAC entry

  Delete from FWD_MACT_IVL if modid is not zero; otherwise, delete from FWD_MACT.

  `bcm_l2_addr_by_struct_delete(unit, l2addr)`

  For details, see the parameters in `bcm_l2_addr_by_struct_get`.
- It is also possible to enable/disable update of existing static-transplantable entry:

  `bcm_switch_control_set(unit, type, arg)`
  - `type` – `bcmSwitchL2ChangeFieldsEnable`
  - `arg`
    - 0: (default value) - Update or replace of existing static-transplantable L2 entries is not allowed.
    - 1: Update or replace of existing static-transplantable L2 entries is allowed. MACT/FID limit is enforced for static-transplantable L2 entries.

NOTE: The `bcmSwitchL2LearnLimitCheckStatic` setting will not apply for static-transplantable entries.

- Get the number of MACT entries for the entire MACT or per VSI/LIF.

  `bcm_l2_addr_count_get(unit, bcm_l2_addr_count_t *count)`
  - `count.flags` – Configuration flags:

Flags	Description
BCM_L2_ADDR_COUNT_TABLE	Count per L2 table
BCM_L2_ADDR_COUNT_VSI	Count per VSI
BCM_L2_ADDR_COUNT_GPORT	Count per LIF

  - `count.vsi` – VSI value. Must be set only when the `BCM_L2_ADDR_COUNT_VSI` flag is provided.
  - `count.gport` – LIF value. Must be set only when the `BCM_L2_ADDR_COUNT_GPORT` is provided.
  - `count.cores_group` – `cores_group` value.
  - `count.count` – Number of entries.

NOTE: To get the number of entries per LIF, the `l2_learn_limit_mode` SOC property must be set to `VLAN_PORT`. To get the total number of entries in the MACT, the VSI and GPORT fields must not be set.

21.14.3 Shell Commands

`l2 add` – Add an L2 entry from the BCM shell.

Example: `l2 add mac=00:00:00:00:44:88 Vlanid=1 PortBitMap=0x1`

`l2 delete` – Delete a MAC entry

Example: `l2 delete mac=00:00:00:00:44:88 Vlanid=1`

21.14.4 Application Reference

Example of basic bridge

- **Type**: CINT reference

- **Path**: `$SDK/src/examples/sand/cint_l2_basic_bridge_with_vlan_editing.c`

Example of static transplant

- **Type**: CINT reference
- **Path**: `$SDK/src/examples/dnx/field/cint_field_static_sa_move.c`

21.15 STG Properties

STP (Spanning Tree Protocol) is a network protocol used to build a logical loop-free topology for Ethernet networks. The basic function of STP is to prevent bridge loops and the broadcast radiation that results from them. Spanning tree also allows a network design to include backup links to provide fault tolerance if an active link fails.

In the actual networks, A VSI (Virtual Switch Instance) is associated with a Spanning Tree Group (STG, also known as Topology-ID). The Spanning Tree Protocol assigns each Spanning Tree Group and Port to a STP state that is used to filter incoming and outgoing packets according to the In/Out-Port and the packet's VLAN.

During initialization stage, a spanning tree group is created with STG ID = 1 and it is set as the default STG on the chip. Thus, VLAN 1, the default VLAN on the chip, is added to the default STG in initialization. All local ports are STP enabled and set with 'FORWARD' state in default STG. When a VLAN is created/destroyed, it also will be added to or removed from the default STG.

NOTE: The default STG can be changed by using `bcm_stg_default_set`. Changing the default STG means changing the protocol state for all ports in the STG.

As the protocol announced, five STP states are supported:

- `BCM_STG_STP_DISABLE` – Spanning Tree protocol is disabled for the port, thus no packets will be filtered by STP.
- `BCM_STG_STP_BLOCK` – Spanning Tree protocol is in 'BLOCK' state. Data packets will be dropped or trapped.
- `BCM_STG_STP_LISTEN` – It is an intermediate state. Data packets will have the same behavior as 'BLOCK' state.
- `BCM_STG_STP_LEARN` – Spanning Tree protocol is in 'LEARN' state. Data packets will be dropped or trapped after learning their source addresses.
- `BCM_STG_STP_FORWARD` – Forwarding state for the port. Spanning Tree protocol is in 'FORWARD' state. Data packets will be forwarded and learned normally.

21.15.1 SOC Properties

None

21.15.2 Configuration Flow

1. Create a Spanning Tree Group by calling:

   `bcm_stg_create(unit, stg_ptr)`

   – `stg_ptr` – The pointer to receive the STG ID.

   It is also possible to create a STG with the given group ID.

   `bcm_stg_create_id(unit, stg)`

   – `stg` – The given STG ID.

NOTE: STG-ID 1 is created by default.

2. Configure (set or change) the STP state for port

   `bcm_stg_stp_set(unit, stg, port, stp_state)`

   – `stg` – The given STG in which the stp state is configured.
   – `port` – The port for whom the stp state is configured.
   – `stp_state` – The stp state to configure for the port in the given STG.
   – The available STP states can be:

STP State	Description
`BCM_STG_STP_DISABLE`	Spanning tree protocol is disabled for the port
`BCM_STG_STP_BLOCK`	Block state for the port
`BCM_STG_STP_LISTEN`	Listen state for the port
`BCM_STG_STP_LEARN`	Learning state for the port
`BCM_STG_STP_FORWARD`	Forwarding state for the port

NOTE: The API assumes symmetric configuration for the incoming and outgoing port.

3. Add a VSI to the given Spanning Tree Group

   `bcm_stg_vlan_add(unit, stg, vid)`

   – `stg` – The given STG ID the VSI is to be added to.
   – `vid` – The VSI ID that will be added to the STG.

NOTE: By default, VSI 1 is mapped to STG 1.

21.15.3 Shell Commands

- STG CReate – Create a Spanning Tree Group
- STG DeSTRoY – Destroy a Spanning Tree Group with the ID. Valid Range is [1,128]
- STG STP Set – Set the STP state for a port in the Spanning Tree Group
- STG STP Get – Get the STP state for a port in the Spanning Tree Group
- STG VLan Add – Add the VID to the Spanning Tree Group
- STG VLan ReMoVe – Remove the VID from the Spanning Tree Group. Use 0xFFFF to remove VSIs in the STG.
- STG DeFaulT Set – Set the default Spanning Tree Group ID for the chip
- STG DeFaulT Get – Get the default Spanning Tree Group ID for the chip
- STG CLear – Clear the STG model and reset it to initial state

21.15.4 Application Reference

Example of STG typical settings

- **Type:** CINT reference
- **Path:** `$SDK/src/examples/sand/cint_stg.c`

21.16 L2 Filters

A set of ingress and egress filters are enabled by default to filter incorrect packets going through the pipeline. The default configuration are done by the Tune module as part of the initialization sequence. The following tables summarize all L2 traps that are enabled by default and the action taken for the packets, qualifying on the traps.

The L2 Ingress filter traps configured on init by default with the actions are summarized in the following table:

L2 Filter Traps Configured on init	Description	Default Action
`bcmRxTrapSameInterface`	Filter packets with same in and out port	Drop
`bcmRxTrapDfltDroppedPacket`	Ingress default drop packet trap	Drop
`bcmRxTrapDfltRedirectToCpuPacket`	Ingress default redirect to CPU trap	Redirect to CPU
`bcmRxTrapStpStateBlock`	Ingress default STP State block trap	Drop, Learning is disabled
`bcmRxTrapStpStateLearn`	Ingress default STP State learn trap	Drop, Learning is enabled
`bcmRxTrapL2Learn0`	Initialize default L2 Learn0 trap	none
`bcmRxTrapL2Learn1`	Initialize default L2 Learn1 trap	Learning is disabled
`bcmRxTrapL2Learn2`	Initialize default L2 Learn2 trap	Learn SA and Drop
`bcmRxTrapL2Learn3`	Initialize default L2 Learn3 trap	Drop
`bcmRxTrapLinkLayerVlanTagDiscard`	Initialize default `LinkLayerVlanTagDiscard` trap	Drop
`bcmRxTrapL2DiscardMacsaFwd`	Initialize default to FWD the packet	Forward
`bcmRxTrapL2DiscardMacsaDrop`	Initialize default to drop the packet	Drop
`bcmRxTrapL2DiscardMacsaTrap`	Initialize default to redirect the packet to CPU	Redirect to CPU
`bcmRxTrapL2DiscardMacsaSnoop`	Initialize default to snoop the packet	Snoop

The L2 ERPP filter traps configured on init by default with the actions are summarized in the following table.

L2 Filter Traps Configured on init	Description	Default Action
`bcmRxTrapEgHairPinFilter`	Filter packets qualifying on the following two conditions: ■ In and out ports are equal; in and out interfaces are ports ■ In and out ports are equal and equal; in and out interfaces are equal and are not ports	Drop

The L2 ETPP filter traps configured on init by default with the actions are summarized in the following table.

L2 Filter Traps Configured on init	Description	Default Action
`bcmRxTrapEgTxPortNotVlanMember`	Drop packets qualifying on PortNotVlanMember filter	Drop
`bcmRxTrapEgTxSplitHorizonFilter`	Drop packets qualifying on Split Horizon filter	Drop
`bcmRxTrapEgTxStpStateFail`	Drop packets qualifying on STP state fail filter	Drop
`bcmRxTrapEgTxProtectionPathUnexpected`	Drop packets qualifying on unexpected protection path filter	Drop

21.16.1 SOC Properties

None

21.16.2 Configuration Flow

Configure traps to different behavior, can be done by the RX Trap APIs. For more information, see Chapter 12, Traps.

21.16.3 Shell Commands

None

21.17 In-AC Wide Data

The ingress LIF database contains LIF entries. Each entry has its own format. Part of the InLIF types may use a format that contains wide (general) data.

The wide data is general data associated with a specific InLIF object.

The data value is controlled by the user and may be used by the ingress ACL as a qualifier.

Typical usage of the general data:
- Extension for the InLIF profile.
- Additional data per InLIF for unexpected user requirements or workarounds.

Wide data is supported for In-AC only (VLAN translation AC LIF and L2VPN/P2P AC). Setting the data itself is done by a dedicated API, `bcm_port_wide_data_set`.

NOTE: Wide data size can be found in Appendix A, Device Family Differences.

The wide data is transferred (for In-AC non-native and native). Eventually, it is visible to the ingress PMF as qualifier.

It is possible to configure an ingress ACL rule that uses the wide data as a qualifier. To do this, use an ACL user-defined qualifier sequence (as described in Chapter 11, Field Processor).

In addition, 8 MSB bits can be used as PMF1 preselector.

(qual_type=bcmFieldQualifyAcInLifWideData).

Wide Data Extension

An option is available to extend the In-AC wide data by using the `bcm_switch_wide_data_extension_add` API.

NOTE: Wide data extension is not supported for Native-AC.

The extension lookup key mode can be one of the following:

- Wide data mode – The lookup key is 18 LSB bits of the In-AC wide data. This is possible only on In-AC objects that include wide data. This is the default mode.
- AC LIF mode – The lookup key is the AC global LIF.

Choosing the key mode is made by the `bcm_switch_control_set` API.

- `type=bcmSwitchWideDataExtKeyMode`
- `arg=bcmSwitchWideDataExtKeyModeByAcLif` or `bcmSwitchWideDataExtKeyModeByWideData`

Both options will do the lookup on the same table and retrieve from it the extended data.

The key mode is device configuration and cannot be changed when the table is not empty.

It is possible to configure an ingress ACL rule that uses the wide data extension result as a qualifier. To do this, use an ACL user-defined qualifier sequence (as described in Chapter 11, Field Processor).

21.17.1 SOC Properties

None

21.17.2 Configuration Flow

Set the wide data extension key mode by API.

`bcm_switch_control_set`

- `type=bcmSwitchWideDataExtKeyMode`
- `arg=bcmSwitchWideDataExtKeyModeByAcLif` or `bcmSwitchWideDataExtKeyModeByWideData`

If not set, the default mode is `bcmSwitchWideDataExtKeyModeByWideData`.

Create an In-AC object. As part of the objection creation indicate wide-data is required:

- For VLAN-Port: Call `bcm_vlan_port_create` with flag `BCM_VLAN_PORT_INGRESS_WIDE`.

NOTE: Wide data is not available for all AC types. For example, native virtual AC-LIFs. In other words, the following flag combinations are not allowed:

```
BCM_VLAN_PORT_INGRESS_WIDE
BCM_VLAN_PORT_VLAN_TRANSLATION
BCM_VLAN_PORT_NATIVE
```

Set the wide data value by using `bcm_port_wide_data_set(unit, gport, flags, data)`

- `gport` – InLIF gport which decode the local lif. (for example, VLAN-Port gport).
- `flags` – `BCM_PORT_WIDE_DATA_INGRESS`.
- `data` – The wide-data value.

Set the wide data extension value using `bcm_switch_wide_data_extension_add (unit, flags, info)`:

- `flags`
  - `BCM_SWITCH_WIDE_DATA_EXTENSION_REPLACE` (replace or modify an entry)
  - `BCM_SWITCH_WIDE_DATA_EXTENSION_GPORT_KEY` – Indicates that the valid key is `gport_key`. If it does not exist, the valid key is `wide_data_key`.
- `info.wide_data_key` – The wide data extension key.

- `info.gport_key` – AC LIF gport, used as the wide data extension key if the relevant flag exists.
- `info.wide_data_result` – The wide data extension result.

Configure ACL rule, which use the wide data as a qualifier.

- Use `bcmFieldQualifyAcInLifWideData` or `bcmFieldQualifyNativeAcInLifWideData` as the qualifier type.

If the wide data extension was configured, configure the ACL rules that use the wide data extension value as a qualifier.

- Use `bcmFieldQualifyAcInLifWideDataExtended` or `bcmFieldQualifyAcInLifWideDataExtendedRaw`

21.17.3 Shell Commands

None

21.17.4 Application Reference

Example of wide data and wide data extension usage:

- **Type:** CINT reference
- **Path:** `$SDK/src/examples/dnx/cint_inlif_wide_data.c`

21.18 VLAN Compression

VLAN compression can be used to reduce the number of <In-Port, VLAN, VLAN> distinct key combinations (and SEM entries), required for the In-AC classification. If a packet VLAN matches a range, the compressed-VLAN value (i.e., the range low value), is used in the match VLAN field used for creating a new AC (`bcm_vlan_port_create` or `bcm_port_match_add`).

Each In-Port can support each one of the global VLAN-ranges (either on inner VLANs or on outer VLANs). Note that the original packet's VLAN is used in the ingress packet processing (for example, VLAN membership filters), except for the construction of the In-AC match key.

21.18.1 SOC Properties

None

21.18.2 Configuration Flow

To configure VLAN compression range per port call.

```
bcm_vlan_translate_action_range_add( int unit,
bcm_gport_t port,
bcm_vlan_t outer_vlan_low, bcm_vlan_t outer_vlan_high,
bcm_vlan_t inner_vlan_low, bm_vlan_t inner_vlan_high,
bcm_vlan_action_set_t *action)
```

- `port` – Incoming port to add vlan range.
- (`outer_vlan_low, outer_vlan_high`) – Specify the outer VLAN range.
- (`inner_vlan_low, inner_vlan_high`) – Specify the inner VLAN range.
- `action` – VLAN compression actions:
  - `action->it_inner = bcmVlanActionCompressed` – For packets with a single inner tag.
  - `action->dt_inner = bcmVlanActionCompressed` – For double-tagged packets.

21.18.3 Shell Commands

None

21.18.4 Application Reference

Example of VLAN compression.

- **Type**: CINT reference
- **Path**: `$SDK/src/examples/sand/cint_port_vlan_range.c`

21.19 MACT IVL

NOTE: The Independent VLAN Learning (IVL) feature is not available in the BCM88690 A0 device.

When referring to MACT, the SDK can configure either the FWD_MACT table (regular MACT) or FWD_MACT_IVL table (dedicated table). The IVL MACT entry FWD and Learn lookup uses keys in the form of {VLAN-tag; FID; MAC}, where the VLAN-tag is the outer VLAN after IVE. For untagged packets, IVE should exist on related LIFs. For more information about IVE, see Chapter 16, VLAN Editing Mechanism.

21.19.1 SOC Properties

None

21.19.2 Configuration Flow

- Create a VSI object.
  See Section 21.9.2, Configuration Flow.
- Set an IVL property per VSI.
  `bcm_vlan_control_vlan_set(unit, vsi, control)`
  `- control.flags2: BCM_VLAN_FLAGS2_IVL`
- Create an IVL Service AC LIF.
  `bcm_vlan_port_create(unit, vlan_port)`
  `- vlan_port.flags: BCM_VLAN_PORT_IVL`
- Associate VPN to VLAN-Port.
  See Section Section 27.5.2, Configuration Flow.
- Apply ingress VLAN editing for ACs or native VLAN editing for tunnels.
  See Chapter 16, VLAN Editing Mechanism.
- Add, delete, or get a MAC entry.
  See Section Section 21.14.2, Configuration Flow

21.19.3 Shell Commands

Use the `l2 add` command to add an IVL MAC entry from the BCM shell.

Example:

```
l2 add mac=00:00:00:00:44:88 vsi=1 PortBitMap=0x1 vid=20
```

Use the `l2 delete` command to delete an IVL MAC entry from the BCM shell.

Example:

```
l2 delete mac=00:00:00:00:44:88 vsi=1 vid=20
```

For more L2 command information, refer to the *Shell Command* user manual.

21.19.4 Application Reference

Example of basic bridge IVL

- **Type**: CINT reference
- **Path**: `$SDK/src/examples/dnx/cint_dnx_mact_ivl.c`

Example of L2 compatible multicast IVL

- **Type**: CINT reference
- **Path**: `$SDK/src/examples/dnx/multicast/cint_dnx_multiast_bridge_ivl.c`

21.20 API Descriptions

21.20.1 TPID Definitions (bcm_switch_tpid_*)

API Name	Highlights
`bcm_switch_tpid_add()`	Add a TPID value to the Global TPID's table
`bcm_switch_tpid_delete()`	Delete a TPID value from the Global TPID's table
`bcm_switch_tpid_get_all()`	Retrieve all the TPIDs that exist in the Global TPID's table
`bcm_switch_tpid_delete_all()`	Delete all the TPIDs from the Global TPID's table

21.20.2 Port Bridging Properties

API Name	Highlights
`fbcm_port_class_set()`, *`bcmPortClassId`*	Associate port with VLAN domain
`bcm_port_class_get()`, *`bcmPortClassId`*	Retrieve port's VLAN domain
`bcm_port_untagged_vlan_set()`	Set port default VLAN
`bcm_port_untagged_vlan_get()`	Get port's default VLAN
`bcm_port_untagged_vlan_get()`	Get port's default VLAN
`bcm_port_control_set()` with type `bcmPortControlEgressFilterDisable`	Per port it is possible to disable egress filters for Unknown type packets.
`bcm_port_control_get()` with type `bcmPortControlEgressFilterDisable`	Retrieve port indication of egress filters for Unknown type packets.
`bcm_vlan_control_port_set()` with type `bcmVlanPortIgnorePktTag`	Port will Ignore if packet is tagged or not
`bcm_vlan_control_port_get()` with type `bcmVlanPortIgnorePktTag`	Get port indication is it ignoring packet tag or not
`bcm_vlan_control_port_set()` with type `bcmVlanTranslateIngressMissDrop`	Assign port default In-AC to be Drop-AC or Initial-AC
`bcm_vlan_control_port_get()` with type `bcmVlanTranslateIngressMissDrop`	Indicate port default In-AC mode (Drop-AC or Initial-AC)

API Name	Highlights
bcm_port_learn_set()	Enable learn per port.
bcm_port_learn_get()	Get learn enabled/disabled per port.

21.20.3 Port x VLAN Properties

API Name	Highlights
bcm_vlan_port_add()	Add port VLAN-membership and TX-tag functionality by a given VLAN and bitmaps of ports. Symmetrical configuration between ingress and egress.
bcm_vlan_gport_add()	Add port VLAN-membership and TX-tag functionality by a given VLAN and port. API supports Asymmetrical configuration and ability to configure per property.
bcm_vlan_gport_get()	To get information about the provided port for the given VLAN. Parameter flags are the same as used in bcm_vlan_gport_add().
bcm_vlan_port_get()	Gets VLAN port membership per VLAN ID call bcm_vlan_port_get. It will return port bitmap of the membered ports and port bitmap of the membered ports which have tx_tag valid. The returned parameter pbmp is a port bitmap of the membered ports. The eturned parameter ubmp is a port bitmap of the membered ports that have tx_tag valid.
bcm_vlan_gport_get_all()	Gets information about the given VLAN ID call. Returned parameter array_max represents the maximum number of ports to be retrieved for the relevant VLAN. Returned parameter gport_array is the array of ports associated with the VLAN. Returned parameter is_untagged indicates, per port, whether it is untagged. Returned parameter array_size is the actual array size (number of ports that are retrieved for the relevant VLAN).
bcm_vlan_list_by_pbmp()	Gets an array of defined VLANs and their port bitmaps call bcm_vlan_list_by_pbmp. The returned VLANs contain at least one of the specified ports from the input port bitmap. Parameter ports is the port bitmap for selection of VLANs. Parameter listp is the list to store all VLANs information. Returned parameter countp is the total number of VLANs.
bcm_vlan_gport_delete()	Removes a single port from the VLAN port membership.
bcm_vlan_port_remove()	Removes ports from the VLAN port membership. Parameter pbmp present the logical ports bitmap containing the relevant ports to be removed.
bcm_vlan_gport_delete_all()	Removes all ports from port membership per given VLAN ID.

21.20.4 Port x VLAN-Tag-Structure

API Name	Highlights
bcm_port_tpid_class_set()	Set VLAN tag classification for a port.
bcm_port_tpid_class_get()	Get VLAN tag classification for a port.

21.20.5 In-AC Wide Data

API Name	Highlights
bcm_port_wide_data_set()	Set the wide data value for a specific lif object (gport).
bcm_port_wide_data_get()	Get the wide data value for a specific lif object (gport).

API Name	Highlights
bcm_switch_wide_data_extension_add()	Add an entry that extends the wide data. ■ wide_data_key: Wide data as configured in the bcm_port_wide_data_set API. ■ wide_data_result: Extended data.
bcm_switch_wide_data_extension_get()	Get a wide data extension entry by its key.
bcm_switch_wide_data_extension_delete()	Delete a wide data extension entry by its key.
bcm_switch_wide_data_extension_traverse()	Traverse over all valid wide data extension entries and call the given callback.

21.20.6 AC VLAN Translation

API Name	Highlights
bcm_vlan_port_create(int unit, bcm_vlan_port_t * vlan_port)	Create a L2 logical port
bcm_vlan_port_destroy(int unit, bcm_gport_t gport)	Destroy a L2 logical port
bcm_vlan_port_find (int unit, bcm_vlan_port_t * vlan_port)	Get/find a layer 2 logical port given the GPORT ID
bcm_port_match_add(int unit, bcm_gport_t port, bcm_port_match_info_t * match_info)	Add a match to an existing port
bcm_port_match_delete(int unit, bcm_gport_t port, bcm_port_match_info_t * match_info)	Remove a match from an existing port
bcm_port_match_delete_all(int unit, bcm_gport_t port, bcm_port_match_info_t * match_info)	Remove all matches from an existing port
bcm_port_match_replace(int unit,bcm_gport_t port, bcm_port_match_info_t * old_match, bcm_port_match_info_t * new_match)	Replace an old match with a new one for an existing port.
bcm_port_match_set(int unit, int size, bcm_gport_t port, bcm_port_match_info_t * match_array)	Assign a set of matches to an existing port
bcm_port_match_multi_get(int unit, bcm_gport_t port, int size, bcm_port_match_info_t * match_array, int *count)	Get match objects that are using the given virtual egress default VLAN port. (vlan_port with subtype of VIRTUAL EGRESS DEFAULT) or VLAN-Port ingress match entries. Size presents the number of match objects to retrieve. It can be 0, then the API only returns the total number of the related match objects but the object contents.

21.20.7 VSI Ethernet Bridging Properties

API Name	Highlights
bcm_vlan_create	Create VSI object, ID supported 1-4K.
bcm_vlan_destroy	Destroy VSI object.
bcm_vswitch_create	Create VSI object, without WITH-ID option.
bcm_vswitch_create_with_id	Create VSI object, WITH-ID option.
bcm_vswitch_destroy	Destroy VSI object.
bcm_vlan_control_vlan_set	Set additional VSI properties. When the MACT limit mode is not per VLAN, aging_cycles is meaningless and is not fetched.
bcm_vlan_control_vlan_get	Get additional VSI properties. When the MACT limit mode is not per VLAN, aging_cycles is meaningless and is not fetched.

21.20.8 MACT Management: Aging

API Name	Highlights
`bcm_l2_age_timer_meta_cycle_set()`	Set time between runs of the age machine. Set the time to 0 in order to disable the automatic aging.
`bcm_l2_age_timer_meta_cycle_get()`	Get time between runs of the age machine
`bcm_switch_control_set()` with the flag `bcmSwitchL2AgeScan`	Trigger a cycle of the age machine. The age of the MAC entries is reduced in the same way as if a meta cycle time had passed. This API can be called only when the automatic aging is disabled (that is, `bcm_l2_age_timer_meta_cycle_set` was set to 0)

21.20.9 MACT Management: Flush Machine

API Name	Highlights
`bcm_l2_traverse()`	Call a user callback on each l2 entry in the MACT.
`bcm_l2_matched_traverse()`	Call a user callback on l2 entries that satisfy the given rule.
`bcm_l2_match_masked_traverse()`	Call a user callback on l2 entries that satisfy the given rule and mask.
`bcm_l2_replace()`	Update payload of all the MACT entries that satisfy the rule to a new port.
`bcm_l2_replace_match()`	Update payload of all the MACT entries that satisfy the rule to the new payload properties.
`bcm_l2_addr_delete_by_port()`	Delete all the entries that have the given port as destination.
`bcm_l2_addr_delete_by_mac()`	Delete all the entries that have the given MAC as a key.
`bcm_l2_addr_delete_by_vlan()`	Delete all the entries that have the given VSI as a key.
`bcm_l2_addr_delete_by_trunk()`	Delete all the entries that have the given trunk as the destination.
`bcm_l2_addr_delete_by_mac_port()`	Delete all the MAC entries that have a given MAC and port.
`bcm_l2_addr_delete_by_vlan_port()`	Delete all the MAC entries that have a given VSI and port.
`bcm_l2_addr_delete_by_vlan_trunk()`	Delete all the MAC entries that have a given VSI and trunk.
`bcm_switch_control_set()` `bcmSwitchTraverseMode`	Update the traverse mode from regular to bulk and back. Use this API the add rules and clear rules in bulk mode.
`bcm_l2_key_dump()`	Print a l2 key to console

21.20.10 MACT Management: L2 Learn

API Name	Highlights
`bcm_switch_control_set` with `bcmSwitchL2LearnMode`	Set learn mode.
`bcm_switch_control_get` with `bcmSwitchL2LearnMode`	Get learn mode.
`bcm_l2_addr_msg_distribute_set()`	Configure the learn events' distribution.
`bcm_l2_addr_msg_distribute_get()`	Get the learn events' distribution. Returns BCM_L2_ADDR_DIST_SET_NO_DISTRIBUTER if there are no distribution queues for the given event.
`bcm_l2_learn_msgs_config_set()`	Configure the header of the DSP messages.
`bcm_l2_learn_msgs_config_get()`	Fetch the header of the DSP messages.
`bcm_l2_addr_register()`	Register a callback for CPU learning.
`bcm_l2_addr_unregister()`	Unregister a CPU learning callback.

API Name	Highlights
bcm_l2_learn_limit_set()	Configure a limit to the number of MAC entries for the whole table, VSI
bcm_l2_learn_limit_get()	Get the configured limits on the MAC table.
bcm_l2_clear()	Clear the L2 callbacks, deinit and init the L2 module.
bcm_l2_learn_action_resolution_set()	Set action resolution for weaker over stronger transplant
bcm_l2_learn_action_resolution_get()	Get the gport action for weaker over stronger transplant

21.20.11 MACT Management: Registration upon MACT Changes

API Name	Highlights
bcm_l2_addr_register()	Register a callback to handle the MAC table's events.

21.20.12 L2 MACT Forwarding

API Name	Highlights
bcm_l2_addr_add	Add a MAC entry to FWD_MACT or FWD_MACT_IVL.
bcm_l2_addr_get	Get a MAC entry from FWD_MACT.
bcm_l2_addr_delete	Delete a MAC entry from FWD_MACT.
bcm_l2_addr_by_struct_get	Get a MAC entry from FWD_MACT or FWD_MACT_IVL.
bcm_l2_addr_by_struct_delete	Delete a MAC entry from FWD_MACT or FWD_MACT_IVL.

21.20.13 STG

API Name	Highlights
bcm_stg_create(int unit, bcm_stg_t *stg_ptr)	Create a spanning tree group.
bcm_stg_create_id(int unit, bcm_stg_t stg)	Create a spanning tree group with the given topology ID.
bcm_stg_destroy(int unit, bcm_stg_t stg)	Destroy the given spanning tree group.
bcm_stg_stp_set(int unit, bcm_stg_t stg, bcm_port_t port, int stp_state)	Set the Spanning Tree Protocol state of a port in the specified STG.
bcm_stg_stp_get(int unit, bcm_stg_t stg, bcm_port_t port, int *stp_state)	Return the Spanning Tree Protocol state of a port in the specified STG.
bcm_stg_vlan_add(int unit, bcm_stg_t stg, bcm_vlan_t vid)	Add a VSI to a specified STG.
bcm_stg_vlan_remove(int unit, bcm_stg_t stg, bcm_vlan_t vid)	Remove a VSI from a STG.
bcm_stg_vlan_remove_all(int unit, bcm_stg_t stg)	Remove all VSIs from a STG.
bcm_stg_vlan_list(int unit, bcm_stg_t stg, bcm_vlan_t **list, int *count);	Generates a list of VSIs in a specified STG.
bcm_stg_vlan_list_destroy(int unit, bcm_vlan_t *list, int count);	Destroy a list returned by bcm_stg_vlan_list. Must be called to release the memory allocated for the list if these VSIs in list are not needed anymore.
bcm_stg_count_get(int unit, int * max_stg)	Return the maximum number of STGs that the hardware can support.
bcm_stg_default_set(int unit, bcm_stg_t stg)	Designate the default STG ID for the chip.

API Name	Highlights
`bcm_stg_default_get(int unit, bcm_stg_t *stg_ptr)`	Return the current default STG ID for the chip.
`bcm_stg_clear(int unit)`	Destroy all STGs and initialize the STG module to its initial configuration.

21.20.14 VLAN Compression

API Name	Highlights
`bcm_vlan_translate_action_range_add()`	Add new VLAN range to a specified port.
`bcm_vlan_translate_action_range_get()`	Get VLAN range for a specified port.
`bcm_vlan_translate_action_range_delete()`	Delete existing VLAN range for a specified port.
`bcm_vlan_translate_action_range_delete_all()`	Delete all existing VLAN ranges for all ports.

Chapter 22: IP Router

22.1 Introduction

This chapter provides further details on IP router (IPv4 and IPv6) unicast and multicast operation and application configuration.

NOTE: The following functions are not discussed in this section:

- L2 functionality. This functionality is required to ramp-up the L3 applications, but it is not in the scope of this section and is explained in Chapter 21, Ethernet Bridge.
- L3 tunnels (MPLS/IP tunnels). For more information about L3 MPLS Tunnels, see Chapter 23, MPLS Label Switch Router. For more information about L3 IP tunnels, see Chapter 27, Q-in-Q Bridging through Chapter 30, IP Tunnel v6 Termination.

22.2 Application Configuration Checklist

When configuring an IP router application, verify that the operations listed in this section are configured.

- Port Routing Properties
- My-MAC Layer 2 Termination
- My-MAC Enhancements (VRRP and Multiple My-MAC)
- ETH-RIF (L3 VSI Interface) and VRF Definition
- L3 Egress Object: Egress-ARP (Next-hop)
- L3 Egress Object: Ingress-FEC
- L3 Egress Object: Protection-FEC
- L3 Egress: ECMP object
- Adding Host Entries
- Adding Route Entries
- Adding IPMC Entries
- Adding vPBR Entries
- L3 Filters
- Performance Improvement

22.3 Port Routing Properties

This section describes the main port routing properties. The information provided here is in addition to port bridging properties. For more information on port bridging properties, see Section 21.4, Port Bridging Properties

Per port, it is possible to decide how to derive the routing interface (ETH-RIF). ETH-RIF information is located in Section 22.6, ETH-RIF (L3 VSI Interface) and VRF Definition. following modes are possible:

- Mode that equates In-Port with VSI and identifying the routing interface (ETH-RIF) with the In-Port (irrespective of its VLAN tags). The decision is per In-Port.
- Mode that equates VLAN-tag with VSI and identifying the routing interface (ETH-RIF) with the VLAN. The decision is per In-Port.
- A routing interface (ETH-RIF) that straddles several LANs, each with its VLAN naming space. In that case, each LAN port constitutes a {VLAN-domain, VLAN-tag(s)} to VSI. VLAN-Domain naming space is configured per In-Port.

22.3.1 SOC Properties

None

22.3.2 Configuration Flow

Derive the routing interface (ETH-RIF):

- According to port—Create a new default In-AC (option2 in default LIF for incoming port). For the exact sequence, see Section 21.4, Port Bridging Properties. In this case, set the VSI field to equal= the required ETH-RIF.
- According to VLAN-tag—Use default AC Initial-AC. In this mode all packets arriving the port, the VSI equals the VLAN. It is also possible to create a new default In-AC (option2 in default LIF for incoming port). For the exact sequence, see Section 21.4, Port Bridging Properties. In this case, use flag Optimized mode.
- According to {VLAN-domain, VLAN-tags}—Follow the steps in Section 21.6.4, Application Reference.

22.3.3 Shell Commands

None

22.3.4 Application Reference

Example of how to derive the routing interface (ETH-RIF) according to port.

- **Type**: CINT reference
- **Path**: `src/examples/sand/cint_sand_utils_l3.c`

22.4 My-MAC Layer 2 Termination

A single MAC address can be assigned per VSI (ETH-RIF). The single MAC address is assumed by SDK to be symmetric at the device (ingress and egress). For a given VSI (ETH-RIF) ID, at the ingress, the MAC address is used to match packet's Ethernet DA to decide on layer 2 termination. At the egress, it is used to derive the packet's Ethernet SA for routing packets.

22.4.1 SOC Properties

None

22.4.2 Configuration Flow

To set the MAC address for a specific VSI (ETH-RIF), see Section 22.6, ETH-RIF (L3 VSI Interface) and VRF Definition.

22.4.3 Shell Commands

None

22.4.4 Application Reference

None

22.5 My-MAC Enhancements (VRRP and Multiple My-MAC)

Two mechanisms allow configuring additional my-MAC rules for L2 termination. The first method is based on hard-logic involving VSI and small TCAM. The second method is to use the EM database of MDB. For both methods, the target of additional My-MAC rules is usually for VRRP but also can be used for additional specific applications that require special My-MAC rules (for example, MPLS multicast reserved address or Port-based termination).

In all cases that run in parallel to a certain packet, considering the protocol (the next layer-type of the packet header is usually determined according to the EtherType), DA, VSI or Port if to terminate the Ethernet header or not. If a successful match is found in one of the methods, My-MAC is set and the ETH-RIF procedure is performed.

22.5.1 My-MAC Based on VSI Table and TCAM

Each VSI is associated with a subset of TCAM entries. If a TCAM-hit occurs and the packet's VSI ID is associated with the TCAM entry, then My-MAC is set. If an incoming packet is classified to VSI, then the packet DA and the Ethernet.Next-Protocol are matched to the TCAM entries. If a match is found between the matched TCAM entry and the VSI, my MAC is asserted, and if ETH-RIF is successful the L2 header is terminated.

NOTE: Only a single match in TCAM is expected according to priority (TCAM entry ID).

Protocol-groups combine multiple protocols into one group. It is used to reduce the amount of TCAM entries per protocol. Up to four protocol-groups are supported. For example, a configuration may configure the four protocol-groups as {{Everything else}, {IPv4}, {}, {IPv6}}. Another example is {{Everything else}, {IPv4, IPv6},{},{MPLS} }. The protocol-group may be masked as part of the TCAM entry. For example, a DA may be further associated with Protocol-group 1 and 3 or with Protocol-group 2 and 3.

TCAM DAs may be used as VRRP VRID addresses and VRID-ID, where VRID-address[47:8]= 00-00-5E-00-01-XX for IPv4, {IPv4,IPv6} or VRID-address[47:8] = 00-00-5E-00-02-XX for IPv6. XX, which indicates the VRID-ID. Each VSI can be associated with one (or none) of the VRIDs.

It is possible to use the method for both generic L2-termination and VRRP. Both may apply for inner (native) and outer (overlay L2 layer) termination.

NOTE: As mentioned above, VRRP termination uses the same TCAM mechanism, and by default, Protocol-Groups consist of: 1 for {IPv4}, and 3 for {IPv6}. If Protocol-Groups 1 and 3 are changed, VRRP native BCM API support stops working.

22.5.2 My-MAC Based on EXEM

To enhance the scale of My-MAC rules, it is possible to use the EXEM tables to configure additional My-MAC rules.

The following applications are supported using EXEM My-MAC:

- Per-VSI – My-MAC based on <VSI, MAC, MAC mask>. This method is similar to TCAM based multiple My-MAC (see Section 22.5.1, My-MAC Based on VSI Table and TCAM) but in this case protocol is not part of the rule.
- Per-Port – My-MAC based on <In-Port, MAC, MAC mask>.
- Per-Port and protocol based on <In-Port, Ethernet-type> where Ethernet-type must be MPLS.
- VRRP – Based on IPvX, VSI, MAC DA prefix, and VRID (the last byte of the MAC address). These DAs may be used as VRRP VRID addresses, where VRID[47:8]= 00-00-5E-00-01-XX for IPv4, and VRID[47:8]= 00-00-5E-00-02-XX for IPv6.

NOTE: The following notes apply:

- EXEM-based VRRP uses the same protocol groups as VSI based VRRP for identifying L3 protocols (group 1 for IPv4 and group 3 for IPv6).
- When adding an entry with the extended option, both EXEM tables (for the outer and native ETH) are updated. (The EXEM-based My-MAC can work with a single EXEM if the MDB profile holds only one of the EXEM databases, but functionally, it will work according to the valid EXEM.)
- It is a device decision to use EXEM per In-Port application or per-VSI. It can't have both port based and VSI based at the same time. On initialization-time it is expected to set EXEM system mode, either In-Port or VSI.

22.5.3 SOC Properties

None

22.5.4 Configuration Flow

Protocol-Groups are configured (required for both methods) by calling:
`bcm_switch_l3_protocol_group_set(unit,flags,group)`

- `flags` – Bitmap of the protocols. Following protocols are supported:

Protocol Flag	Protocol-Name
BCM_SWITCH_L3_PROTOCOL_GROUP_IPV4	IPv4
BCM_SWITCH_L3_PROTOCOL_GROUP_IPV6	IPv6
BCM_SWITCH_L3_PROTOCOL_GROUP_ARP	ARP
BCM_SWITCH_L3_PROTOCOL_GROUP_MPLS	MPLS

- `group` – Protocol-group.

  By default, init groups 1 and 3 are reserved for implementing VRRP with IPv6 or IPv4. Modifying their contents would prevent VRRP to work (for both modes).

  Before changing a certain Protocol-group, it is expected that first all VSIs associated with the previous protocol-group protocol values must be dissociated, the new protocol-group protocol values are defined, and then the VSIs are re-associated to the new Protocol-group. Note that if the Protocol-group 1,3 are modified then IPv4 and IPv6 must be re-associated with a new group to use them with VRRP.

22.5.4.1 My-MAC Based on TCAM+VSI

1. To configure an L2 generic termination address entry, call:
   `bcm_l2_station_add(unit,station_id,station)`
   - `station_id` – Returned handle ID to be used for get/destory.
   - `flags` – Refer to a single Protocol-group (as defined in item1). They may also straddle multiple Protocol-groups if required (for example, protocol-group 1 and 2 or protocol-group 1 and 3). Following flags are supported (similar to item1):

Protocol Flag	Protocol-Name
BCM_L2_STATION_NONE	None
BCM_L2_STATION_IPV4	IPv4
BCM_L2_STATION_IPV6	IPv6
BCM_L2_STATION_MPLS	MPLS
BCM_L2_STATION_MIM	MIM
BCM_L2_STATION_FCOE	FCOE
BCM_L2_STATION_ARP_RARP	ARP

   - `station.vlan` – VSI-ID.
   - `station.vlan_mask` – Must be set to 0xFFFF
   - `station.dst_mac` – DA MAC address to consider
   - `station.dst_mac_mask` – Set if to consider or not per bit

NOTE: It is expected to create a VSI (for example, `bcm_vswitch_create_with_id`) before using the API.

2. To configure a VRRP address, call: `bcm_l3_vrrp_config_add(unir,flags,vlan,vrid)`
   - `flags` – Either `BCM_L3_VRRP_IPV4` or `BCM_L3_VRRP_IPV6`.
   - `vlan` – VSI-ID
   - `vrid` – Possible values range between 0 and 255.

NOTE: It is expected to create a VSI (for example, `bcm_vswitch_create_with_id`) before using the API.

22.5.4.2 My-MAC Based on EXEM

1. To configure non-VRRP (l2_station) system mode (VSI, In-Port, or In-Port + protocol), use:
   `bcm_switch_control_set(unit, bcmSwitchL2StationExtendedMode, arg)`
   – `arg` – For VSI set 0 (default), for In-Port set 1. For In-Port + Protocol set 2.
   It is expected to configure the mode on initialization-time.
2. To configure a non-VRRP L2 generic termination address, call:
   `bcm_l2_station_add(unit,station_id,station)`
   – `station.flags` – Must set `BCM_L2_STATION_EXTENDED` (indicates usage of EXEM table).
   The flags in the following table are supported based on the system mode.

Mode	Supported Flags
VSI	`BCM_L2_STATION_EXTENDED`
SRC port	`BCM_L2_STATION_EXTENDED`
SRC port + protocol (ignore MAC)	`BCM_L2_STATION_EXTENDED, BCM_L2_STATION_MPLS`

   To indicate the protocol type, use the `BCM_L2_STATION_MPLS` flag (only MPLS protocol is supported).
   – `station_id` – Returned handle ID to be used for get or destroy.
   – `station.vlan` – VSI ID for VSI based My-MAC. Should be set to 0 when working with port based My-MAC.
   – `station.vlan_mask` – Must be a value other than 0 for VSI based my-MAC. Should be set to 0 when working with port based My-MAC.
   – `station.dst_mac` – DA MAC address to consider
   – `station.dst_mac_mask` – Set if to consider or not per bit
   – `station.src_port` – In-port for working with In-Port My-MAC. Should be set to 0 when working with VSI based My-MAC.
   – `station.src_port_mask` – Must be a value other than 0 for In-Port My-MAC. Should be set to 0 when working with VSI based My-MAC.

NOTE:

- The `vlan_mask` and `src_port_mask` fields function as indications and not as actual masks. They differentiate between the VSI and the SRC port system modes.
- It is expected to create a VSI (for example, `bcm_vswitch_create_with_id`) before using the API.

3. To configure a VRRP address, call: `bcm_l3_vrrp_config_add(unit,flags, vlan, vrid);`
   – `flags` – Either `BCM_L3_VRRP_IPV4` or `BCM_L3_VRRP_IPV6`, along with `BCM_L3_VRRP_EXTENDED`
   – `vlan` – VSI-ID
   – `vrid`

NOTE: It is expected to create a VSI (for example, `bcm_vswitch_create_with_id`) before using the API.

22.5.5 Shell Commands

None

22.5.6 Application Reference

An example how to add, get, and delete VRRP rules.

- **Type:** CINT reference
- **Path:** `src/examples/dnx/l3/cint_dnx_l3_vrrp.c`

An example how to add, get, and delete multiple My-MAC rules.

- **Type**: CINT reference
- **Path**: `src/examples/sand/cint_multiple_my_mac_term.c`

22.6 ETH-RIF (L3 VSI Interface) and VRF Definition

A VSI L3 routing interface (ETH-RIF) has both ingress and egress properties (In-ETH-RIF and Out-ETH-RIF), it's a system object that has to be synced over all devices and its value is always equal to the VSI.

At the ingress, an ETH-RIF is an interface from VSI and may be achieved by the first MAC Layer 2 termination (Link-layer) or second MAC Layer 2 termination (Native Layer 2 termination). The settings of In-ETH-RIF are shared between the first and second MAC termination (for example, IPv4 UC routing on certain ETH-RIF will influence both cases of first and second MAC termination). In-ETH-RIF table is shared with the L2 VSI table which means on the same entry some L2 and L3 settings exist.

At the egress, an ETH-RIF is provided from the ingress pipeline (by system-headers) for simple routing or from the ARP interface for an L3 tunnel (for example, MPLS, IP tunnels). Similar to the ingress configuration, the ETH-RIF properties for the case of Native and Link ETH layers are shared.

The number of VSIs that can be routed (ETH-RIFs) differs per device and MDB profile. The range values of ETH-RIFs is between 1 and rif_id_max. ETH-RIF is also considered to be a Global-LIF (Ingress and Egress) and requires to allocate and reserved: Local OutLIF, Global-InLIF and Global OutLIF. Thus, the ETH-RIFs range IDs are part of Global-LIF range IDs. Meaning, once device is up, the number of Global-LIF IDs is split into two ranges (both Ingress and Egress):

- [1-rif_id_max] – Reserve only for ETH-RIFs
- [rif_id_max+1-<Max Global-LIF ID>] – Reserve for other Global-LIF IDs

Also note that ETH-RIF 0 is not valid and cannot be used. It is expected by the user to manage the RIF-IDs over the system.

The following settings are possible to configure per ETH-RIF:

- Single MAC address for Layer 2 Termination/Encapsulation, see more explanation in Section 22.4, My-MAC Layer 2 Termination. Note: The MAC address is assumed to be symmetric between Ingress and Egress device.
- Multiple My-MAC termination: If additional My-MAC addresses are required, per VSI it is possible to define a set of DAs. Mainly used for VRRP application. See more information in Section 22.21.1, My-MAC Enhancements.
- Routing enablers: Enable/Disable routing per Protocol: IPv4/6 UC/MC, MPLS. If the incoming packet is to be L2 terminated and the routing interface is disabled for the next header routing protocol:
  - A trap will be generated (bcmRxTrapMyMac*)
  - The packet will be bridged.

  Other protocols (such as CFM, ARP, and so on) have default configuration and cannot be changed. By default upon creation of ETH-RIF, all routing protocols are enabled.
- VRF – Classifying incoming packet to VRF. The VRF is used as part of the key in the DIP/SIP forwarding/RPF/vPBR lookups.
- Enable/Disable Unicast Reverse Forwarding Path (RPF) and define its mode (Strict or Loose).

  By default, upon creation of ETH-RIF, RPF is disabled.

- Enable/Disable Public Routing, influence on the forwarding lookup scheme.

  By default, upon creation of ETH-RIF, Public routing is disabled.
- QoS-Profile – Indicate the QoS profiles that define the PHB and Remark functionality at the ingress and remark functionality at the egress. For more information, see Chapter 10, QoS.

  By default, upon creation of ETH-RIF, Ingress QoS profile is 0.
- QoS-Model (PHB, Remark, TTL) – Indicate whether the ETH-RIF Remark/PHB model is PIPE, Short-Pipe, or Uniform.
- QOS-Model (ECN) – Indicate whether or not Ingress-RIF is eligible for ECN. Once a RIF is eligible for ECN and a packet is IPv4oETH, ECN applications will be set over the pipeline according to the ECN bits in the IPv4 header. For more information see Section 10.6, Explicit Congestion Notification.
- TTL-Model – Indicate if ETH-RIF TTL model is Pipe, Short-Pipe or Uniform.
- TTL Scope – Configure the TTL threshold per Out-RIF. If a packet has a TTL lower than the TTL threshold for the given RIF, the packet will be filtered.
- MTU – Provide the MTU value for Egress MTU filtering, see more explanation in Chapter 12, Traps.

  By default, upon creation of ETH-RIF, MTU filter is disabled.
- Statistics – In the egress, it is possible to set statistic-ID and profile for Out ETH-RIF counting. Upon creation of ETH-RIF, it is required to set or not to set statistics on the ETH-RIF. It is not possible to update a certain ETH-RIF from non-statistics to statistics or from statistics to non-statistics.

NOTE: Routing enablers, RPF, Public routing properties combinations are shared between all ETH-RIFs.

ETH-RIF is a system object and its configuration in both Ingress and Egress are symmetric and configured by the same API call.

22.6.1 SOC Properties

The SOC property `rif_id_max` indicates the value for the maximal ETH-RIF ID in the system.

22.6.2 Configuration Flow

1. The creation of ETH-RIF and its main interface properties is done by calling the API: `bcm_l3_intf_create(unit, intf);`
   - `intf.l3a_intf_id` – The API will create a new ETH-RIF ID which is set by this field, must be non 0.
   - `intf.l3a_vid` – The VSI associated with the L3 interface. Must be set the same value as `l3a_intf_id`.
   - `intf.l3a_vrf` – VRF-ID
   - `intf.l3a_ttl` – Configure the TTL threshold on the RIF. By default, TTL threshold filtering is not applied unless specified.

NOTE: TTL = 0 and TTL = 1 are separate filters, which are applied by default.

   - `intf.dscp_qos.qos_map_id` – Egress QoS profile ID
   - `intf.ingress_qos_model` – Set Ingress QoS-Model (PHB, Remark) and TTL-Model for ETH-RIF. For more information, see Section 10.3, Ingress PHB/Remarking. In addition, indicate whether the RIF is ECN eligible (by setting `ingress_ecn` to `bcmQosIngressEcnModelEligible`).

– `intf.l3a_mac_addr` – ETH-RIF MAC address (My-MAC). API configures the MAC address symmetric in both Ingress and egress device. For more information, see Section 22.4, My-MAC Layer 2 Termination.

– `intf.l3a_flags`:

Flag Name	Description
`BCM_L3_WITH_ID`	Must be set. The interface ID is provided by the user.
`BCM_L3_REPLACE`	Override existing entry

– `intf.l3a_flags2`:

Flag Name	Description
`BCM_L3_FLAGS2_EGRESS_STAT_ENABLE`	Indicates Egress statistics enabled. **NOTE:** In this API statistics enable cannot be replaced during an update API call (meaning that if set this flag cannot be unset or if unset the flag cannot be set). For more information regarding the creation of statistics interfaces and counting, see Chapter 15, PP Statistics Generation

NOTE: For a replace call, all ETH-RIF properties that are defined by other APIs (for example, `bcm_l3_ingress_create`) are preserved.

2. Some of the ingress ETH-RIF properties are set by: `bcm_l3_ingress_create(unit, ing_intf, &l3if.l3a_intf_id)`

– `l3if.l3a_intf_id` – The ETH-RIF ID.

– `l3if.urpf_mode` – Unicast RPF modes. The following three modes are supported:

- `bcmL3IngressUrpfDisable`
- `bcmL3IngressUrpfLoose`
- `bcmL3IngressUrpfStrict`

It is possible to configure the URPF mode for IPv4 packets and for IPv6 packets, separately, using one of the following:

- `bcmL3IngressUrpfV4DisableV6Loose`
- `bcmL3IngressUrpfV4DisableV6Strict`
- `bcmL3IngressUrpfV4LooseV6Disable`
- `bcmL3IngressUrpfV4LooseV6Strict`
- `bcmL3IngressUrpfV4StrictV6Disable`
- `bcmL3IngressUrpfV4StrictV6Loose`

NOTE: Not supported for BCM88690 and BCM88480.

– `l3if.hash_layers_disable` – Used to remove selected protocol from participating in the load balancing key generation

NOTE: This field is not supported by the BCM88690, BCM88800, or BCM88480.

– `l3if.qos_map_id` – Ingress QoS profile ID

– `l3if.flags`:

Flag Name	Description
`BCM_L3_INGRESS_WITH_ID`	Must be set
`BCM_L3_INGRESS_GLOBAL_ROUTE`	Enable Public routing. Must be set if public forwarding lookups are expected
`BCM_L3_INGRESS_ROUTE_DISABLE_MPLS`	Route enablers: disable routing on MPLS packets (enable if this flag is omitted).

Flag Name	Description
BCM_L3_INGRESS_ROUTE_DISABLE_IP4_UCAST	Route enablers: disable routing on IPv4 UC packets (enable if this flag is omitted).
BCM_L3_INGRESS_ROUTE_DISABLE_IP6_UCAST	Route enablers: disable routing on IPv6 UC packets (enable if this flag is omitted).
BCM_L3_INGRESS_ROUTE_DISABLE_IP4_MCAST	Route enablers: disable routing on IPv4 MC packets. Use the IPv4 MC L2 flows for IPv4 MC packet (IPv4 MC L3 flow will be used if this flag is omitted) .
BCM_L3_INGRESS_ROUTE_DISABLE_IP6_MCAST	Route enablers: disable routing on IPv6 MC packets. Use the L2 IPv6 MC flow for IPv6 MC packet (IPv6 MC L3 flow will be used if this flag is omitted).
BCM_L3_INGRESS_L3_MCAST_L2	Use the v4MC L2 flow for IPv4 MC packets. This flag must be used with the BCM_L3_INGRESS_ROUTE_DISABLE_IP4_MCAST flag to take effect. If this flag is omitted and the BCM_L3_INGRESS_ROUTE_DISABLE_IP4_MCAST flag is set, the IPMC L2 bridge forwarding flow will be selected.
BCM_L3_INGRESS_IP6_L3_MCAST_L2	Use the v6MC L2 flow for IPv6 MC packets. This flag must be used with the BCM_L3_INGRESS_ROUTE_DISABLE_IP6_MCAST flag to take effect. If this flag is omitted and the BCM_L3_INGRESS_ROUTE_DISABLE_IP6_MCAST flag is set, the IPMC L2 bridge forwarding flow will be selected.
BCM_L3_INGRESS_IPMC_BRIDGE_FALLBACK	Enable the bridge fallback option of the RIF. For more information, see Section 22.18, Bridge Fallback.

NOTE: The following notes apply:

- It is expected to create an ETH-RIF ID by `bcm_l3_intf_create` before calling this API.
- Although the API is called `_create`, the API is working as a `_set` API (no object creation is done).
- Detailed information on the IPMC flows can be found in the IPMC subsection.

3. Set ETH-RIF MTU value by calling: `bcm_rx_mtu_set(unit, mtu_config);`
   - `mtu_config.flags` = MTU_RX_MTU_RIF
   - `mtu_config.intf` = ETH-RIF ID (`l3a_intf_id` as received from `bcm_l3_intf_create`)
   - For other values, see Chapter 12, Traps.
4. Overwrite the forwarding or mirroring ETPP action. See Chapter 12, Traps.

22.6.3 Shell Commands

L3 interface

- `l3 intf add`
  Create an L3 interface entry by providing its ID and destination
- `l3 intf delete`
  Delete an existing L3 interface by providing its ID.
- `l3 intf update`
  Update the destination of an L3 interface
- `l3 intf get`
  Find the destination of an existing L3 interface

22.6.4 Application Reference

TBD

22.7 L3 Egress Object: Egress-ARP (Next-Hop)

L3 egress object in SDK is composed of two objects: FEC and ARP. This section will explain how to create an ARP-ID (Egress object) and set its properties. For more information on how to create FEC object (Ingress object) see Section 22.8, L3 Egress Object: Ingress-FEC. The creation of ARP is done by `bcm_l3_egress_create` API. The API is also used to create a FEC object. Unlike previous devices, SDK does not support creating FEC and ARP on the same API call. According to API inputs, SDK will create either FEC or ARP (but not both).

At the egress device, an ARP object is created and preset some of the properties of the next-hop router. ARP is located at the egress encapsulation database (EEDB) and considered to be a Global-LIF. ARP is required to be created for any Router Layer or Tunneling Layer, that requires to construct the Ethernet (i.e. link layer) header.

ARP can be created for the outer-link layer or native link-layer. The decision must be done upon creation.

The following settings are possible to configure per ARP:

- DA MAC-Address – Ethernet's DA upon packet transmission.
- ETH-RIF – Optional. Usually used for L3 (MPLS/IP) tunnels since in this case ETH-RIF is not provided directly from the Ingress (UC) or Egress databases (MC). Used to derive SA MAC-Address.
- SA MAC-Address – Optional. In the typical case SA MAC-Address is derived directly from the ETH-RIF. If multiple SA MAC-Address are required on the same ETH-RIF (for example, for Dual-homing) then it is possible to decide the SA MAC-address according to the ARP entry itself.
- VLAN translation properties – Optional. It is optional to derive VLAN-translation information directly from the ARP entry and not from additional ESEM entry or ESEM default properties. If it is used, the following configuration available: VLAN-action profile, Outer-VLAN and Inner-VLAN. For more information about VLAN translation properties, see Chapter 16, VLAN Editing Mechanism.

  If this property is used, the VLAN PCP and DEI fields can be set and mapped if the QoS model is PIPE.

  Note: The decision of including VLAN-translation properties as part of ARP entry, must be decided on creation and cannot be changed upon replace.
- QoS Profile – Indicate the QoS Remark profile. For more information, see Chapter 10, QoS.
- MTU: Optional – Provide the MTU value for Egress MTU filtering, see more explanation in Chapter 12, Traps.
- Action-profile – Possible to overwrite the ETPP forwarding/mirror action. See more information in Chapter 12, Traps.

Some of the properties of ARP can be replaced. Unless explicitly mentioned in Section 22.7.2, Configuration Flow that replacement is possible, it is assumed the property cannot be replaced.

22.7.1 SOC Properties

None

22.7.2 Configuration Flow

To create an egress object, Egress ARP, call: `bcm_l3_egress_create(unit, allocation_flags, &l3eg, NULL)`

- `allocation_flags`:

Allocation Flags	Description
`BCM_L3_EGRESS_ONLY`	Must be set in order for API to create an ARP object.
`BCM_L3_REPLACE`	Can be used to replace ARP properties
`BCM_L3_KEEP_DSTMAC`, `BCM_L3_KEEP_VLAN`	Do not edit ARP, only update next pointer of InLIF. Those fields can be used only upon replace.
`BCM_L3_NATIVE_ENCAP`	If ARP is required for Native Ethernet layer (for more information, see Section 22.7, L3 Egress Object: Egress-ARP (Next-Hop).

- `l3eg.encap_id` – The allocated ARP-ID. If the input of param is non 0, the API will create the object according to the provided ID. The API returns ARP-ID as intf object of type LIF (`BCM_L3_ITF_TYPE_LIF`).
- `l3eg.mac_addr` – Next hop MAC address. This field is replaceable.
- `l3eg.vlan` – The provided ETH-RIF ID. This field is replaceable as long it was set to a valid RIF on creation.
- `l3eg.src_mac_addr` – The source MAC address (used if the `BCM_L3_FLAGS2_SRC_MAC` is set). This field is replaceable.
- `l3eg.qos_map_id` – The egress remark QoS profile. Must be created by QoS API. This field is replaceable.
- `l3eg.egress_qos_model` – The QoS and TTL model. For details, see Chapter 10, QoS.
- `l3egr.intf` – Tunnel that should be pointing on the created ARP. In case the tunnel is an MPLS tunnel pointing to another tunnel, the ARP is attached to the last tunnel in the linked list. This parameter is not returned by the `bcm_l3_egress_get` API. This field is replaceable.
- `l3egr.vlan_port_id` – ARP entry can be linked to a VLAN port entry for VLAN translation information.
- `l3eg.flags`
  - `BCM_L3_NATIVE_ENCAP` – ARP is required for Native Ethernet layer.
- `l3eg.flags2`

Flags2	Description
`BCM_L3_FLAGS2_VLAN_TRANSLATION`	If the entry has also VLAN-translation properties. Those properties can be set by the VLAN translation API (refer to Chapter 16, VLAN Editing Mechanism), by encoding the interface `l3eg.encap_id` as `BCM_GPORT_VLAN_PORT_ID` of subtype `BCM_GPORT_SUB_TYPE_VLAN_TRANSLATION`. For details on the available GPORT and interface MACROs refer to Section 3, L2 and L3 Ports (GPORTs).
`BCM_L3_FLAGS2_VLAN_TRANSLATION_TWO_VLAN_TAGS`	If VLAN translation is involved, two VLAN tags are required. Must be set together with `BCM_L3_FLAGS2_VLAN_TRANSLATION`. In this case ARP does not support statistics.
`BCM_L3_FLAGS2_SRC_MAC`	If set, ARP will include SA MAC-address information.
`BCM_L3_FLAGS2_EGRESS_STAT_ENABLE`	Indicates Egress statistics enabled. For more information regarding the creation of statistics interfaces and counting, see Chapter 15, PP Statistics Generation.

- `l3eg.encap_access` – Specify from which encapsulation stage the ARP encapsulation procedure will be done. Supported `encap_eccess` options are `bcmEncapAccessAc` and `bcmEncapAccessInvalid` (default). The `bcmEncapAccessAc` option cannot be used with the flags `BCM_L3_NATIVE_ENCAP` or `BCM_L3_FLAGS2_VLAN_TRANSLATION`.

NOTE: By default, ARP (not ARP+AC) is configured in seventh EEDB phase (`bcmEncapAccessArp`).

- Set ARP MTU value by calling: `bcm_rx_mtu_set(unit, mtu_config)`
  - `mtu_config.flags` = MTU_RX_MTU_LIF
  - `mtu_config.gport` = ARP-ID as a gport encoded.

NOTE: `l3eg.encap_id` is received from `bcm_l3_egress_create` encoded as intf. User needs to apply the macro `BCM_L3_ITF_LIF_TO_GPORT_TUNNEL` to make it gport encoded.

For the other parameters, see Chapter 12, Traps.

- Overwrite the forwarding/mirroring ETPP action. For more information, see Chapter 12, Traps.
- If ARP contains VLAN-Translation information (ARP + AC object, flags `BCM_L3_FLAGS2_VLAN_TRANSLATION`), the following API can be used for setting VLAN-Translation properties:

  `bcm_vlan_port_translation_set(unit, port_translation);`
  - `port_translation.gport` – ARP object ID of type tunnel. To make it gport encoded, the macro `BCM_L3_ITF_LIF_TO_GPORT_TUNNEL` needs to be applied.
  - For additional properties, see Chapter 16, VLAN Editing Mechanism.

NOTE: By default an ARP+AC object supports only one VLAN tag addition. If two VLAN tags are required (`port_translation.new_inner_vlan != 0`), use `BCM_L3_FLAGS2_VLAN_TRANSLATION_TWO_VLAN_TAGS` on creation.

22.7.3 Shell Commands

None

22.7.4 Application Reference

TBD

22.8 L3 Egress Object: Ingress-FEC

L3 egress object in SDK is composed of two objects: FEC and ARP. This section will explain how to create a FEC-ID (Ingress object) and set its properties. For more information about how to create ARP object (Egress object) see Section 22.7, L3 Egress Object: Egress-ARP (Next-Hop). The creation of FEC is done by `bcm_l3_egress_create` API. The API is also used to create an ARP object. Unlike previous devices, SDK does not support creating FEC and ARP on the same API call. According to API inputs, SDK will create either FEC or ARP (but not both).

The SDK allows FEC object to be:

- A single-object pointed directly from the forwarding tables
- Part of 1:1 protection group, in this case protected FECs corresponds to a pair of consecutive FECs. Different API sequence calls must be done to achieve that.
- Part of ECMP group of single-object FECs or ECMP group of protection-FECs.
- Part of hierarchical structure, that is, an egress object (FEC/ECMP) can point to this FEC and/or this FEC can point to the next hierarchical FEC/ECMP.

Upon FEC creation, it is mandatory to specify the hierarchical level of FEC and if the FEC is Protection-FEC or single-FEC. It is not possible to move between Protection-FEC to single-FEC (and vice versa) after creation as well as changing its hierarchical level.

For FEC allocation ID information, see Chapter 9, FEC and ECMP Management.

By hardware, FECs are ordered in pairs (Super-FEC). The control of the SDK API is per single FEC and ignores the fact that the FEC is part of a pair. The SDK internally manages the mutual settings of a pair and verifies the allocated ID is aligned to hardware expectations. For the case of Protection-FEC, the object corresponds to a pair of consecutive FECs. In this case the sequence API calls and allocated IDs are different (see Section 22.8, L3 Egress Object: Ingress-FEC).

Hierarchical TM flow (HTM)

In the Weak TM Flow use-case, a packet may be mapped to a certain service or a QoS level provided by a TM flow. An example for usage is in VPLS applications where the PWE or MPLS tunnel is protected or in routing that has a VC label service on top of the MPLS ECMP (or protection) network. Those cases should maintain a certain service (PWE) and QOS level (VC label) provided by a TM flow and not according to the last FEC destination result.

HTM allows setting the TM flow on the first or second hierarchy FEC entry to be of type *Weak TM Flow*. In that case, the TM flow is resolved according to the first *Weak TM Flow* FEC and not according to the last FEC entry.

The following settings are possible to configure per super-FEC (a pair of FECs):

Super-FEC type: Indicates whether the FEC pair is with or without protection and with or without statistics. The Super-FEC type is relevant for two consecutive FECs: an even and an odd FEC ID and cannot be updated while one of these FECs exists.

The following settings are possible to configure per FEC:

- Destination – Can be physical destination (trunk, multicast, system-port, VOQ) or the next hierarchical FEC/ECMP level.
- Interface 1 – First Out-Global-LIF pointer, which can be ETH-RIF (for routing) or L3 tunnel-interface (for example, MPLS/IP tunnels).
- Interface 2 – Second Out-Global-LIF pointer, can be ARP-pointer (for routing) or another L3 tunnel-interface.

NOTE: If both interfaces (1 and 2) are valid for the FEC entry, interface 1 must be ETH-RIF. If two interfaces of LIF type are required, it is expected to create additional hierarchy level FEC (one with the first interface, the other with the second interface).

- MPLS command PHP- FEC can result in MPLS PHP. Egress remark QoS profile is available when QoS model is PIPE.
- MPLS command SWAP/Push – FEC can result in MPLS Push or Swap. In this case, a new label value is provided. The push or swap operation depends onwhether the BCM8867X Compatibility mode (IOP) is enabled. If IOP mode is disabled, it depends on the forwarding layer. If the forwarding layer is MPLS, the operation is Swap; otherwise it is Push. When IOP mode is enabled, the MPLS command can be Push or Swap regardless of the forwarding layer. In this case, no EEDB resource is consumed, and MPLS header information is provided directly by the system headers.
- For MPLS LSR and LER applications, see Chapter 23, MPLS Label Switch Router.

NOTE: When configuring MPLS command through FEC, TTL, and QoS models are PIPE.

- Multicast RPF mode: In order to use multicast RPF, a MC FEC is required. MC FEC is used in forwarding to point to a MC group. For RPF, it includes the information of MC RPF mode: None, Explicit or SIP-base. For IPMC purposes it is also possible to skip FEC and directly point to multicast group but then RPF is not supported. It is possible to configure `NO_RPF` mode which is used to disable UC RPF filtering for MC packets.

22.8.1 SOC Properties

None

22.8.2 Configuration Flow

Get the legal FEC ID ranges (required only when `BCM_L3_WITH_ID` is used in `bcm_l3_egress_create`), call: `bcm_switch_fec_property_get(unit, &fec_config)`.

- `fec_config.flags`:
  - `BCM_SWITCH_FEC_PROPERTY_1ST_HIERARCHY`
  - `BCM_SWITCH_FEC_PROPERTY_2ND_HIERARCHY`
  - `BCM_SWITCH_FEC_PROPERTY_3RD_HIERARCHY`
- `fec_config.start` – A return value. The first FEC ID in the range of the requested hierarchy.
- `fec_config.end` – A return value. The last FEC ID in the range of the requested hierarchy.

To create an egress object ingress FEC, call `bcm_l3_egress_create(unit, allocation_flags, &l3eg, &l3egid)`

- `allocation_flags`:

Allocation Flags	Description
`BCM_L3_INGRESS_ONLY`	Must be set for the API to create a FEC object.
`BCM_L3_WITH_ID`	If present, determines the given FEC value
`BCM_L3_REPLACE`	Can be used to replace FEC properties. Cannot be used to update Super-FEC type.

- `l3egid` – The allocated FEC-ID. For WITH-ID, must be set with the required FEC-ID.
- `l3eg.port` – The destination. Can be physical gport (trunk gport, mcast gport, system-port gport, etc…) or next hierarchical FEC/ECMP. For the latest, it is expected to be encoded gport as `BCM_GPORT_FORWARD_PORT_SET(gport, fec/ecmp_id)`.

  Encode the system port into a gport using `BCM_GPORT_SYSTEM_PORT_ID_SET` (and not Local-Port, Mod-port, and so on) to increase FEC insertion performance.
- `l3eg.intf` – The first Out-Global-LIF. Must be encoded as an intf object (`BCM_L3_ITF_TYPE_RIF or BCM_L3_ITF_TYPE_LIF`).
- `l3eg.encap_id` – The second Out-Global-LIF. Must be encoded as an intf object (`BCM_L3_ITF_TYPE_LIF`).
- `l3eg.mpls_action` – For FEC for MPLS application, the MPLS action is part of the FEC result:
  - `BCM_MPLS_EGRESS_ACTION_PHP` – The action is PHP.
  - `BCM_MPLS_EGRESS_ACTION_PUSH_OR_SWAP` – The action is Push or SWAP (depending on the forwarding layer). This action cannot be used by BCM8867X Compatibility mode (IOP).
  - `BCM_MPLS_EGRESS_ACTION_SWAP` and `BCM_MPLS_EGRESS_ACTION_PUSH`: Can be used by BCM8867X Compatibility mode (IOP) only.
- `l3eg.qos_map_id` – Egress remark QoS profile if `l3eg.mpls_action` is PHP or SWAP (for additional SWAP information, see Section 23.7, Remark Profile for SWAP Actions Coming Out of Ingress (ILM, FEC)).
- `l3eg.mpls_label` – MPLS out label if the `l3eg.mpls_action` is `SWAP`
- `l3eg.flags`:
  - `BCM_L3_2ND_HIERARCHY` or `BCM_L3_3RD_HIERARCHY` or no flags (first hierarchy level) – Decide FEC Hierarchical level.
  - `BCM_L3_MC_RPF_EXPLICIT` or `BCM_L3_MC_RPF_SIP_BASE` or no flags (disable) – Decide Multicast RPF mode.
- `l3eg.hierarchical_gport`

- `l3eg.flags`:

Flags	Description
`BCM_L3_2ND_HIERARCHY`, `BCM_L3_3RD_HIERARCHY`	—
`BCM_L3_MC_RPF_EXPLICIT_ECMP`, `BCM_L3_MC_RPF_EXPLICIT`, `BCM_L3_MC_RPF_SIP_BASE`, `BCM_L3_MC_NO_RPF`	Decide multicast RPF mode. If none are set then it is reserved (disabled). For more information, see Section 22.16, Multicast RPF.
`BCM_L3_ENCAP_SPACE_OPTIMIZED`	Used in Interop mode. For more information, See Chapter 5.9, L3 Egress Ingress-FEC Entry: OutLIF Pointer as EEI.
`BCM_L3_HIT_CLEAR`	Upon `get()`, clear hit-bit
`BCM_L3_HIT`	Upon `get()`, indicates an FEC entry is hit (otherwise an FEC entry is not hit)

- `l3eg.flags2`:

Flags	Description
`BCM_L3_FLAGS2_ECMP_RANGE_OVERLAP`	The FEC ID is overlapping the ECMP range. The flag should be used only when the FEC is a member of an ECMP group. See more information in FEC/ECMP management section.
`BCM_L3_FLAGS2_SKIP_HIT_CLEAR`	Upon `create()`, indicates whether to skip clearing the hit bit (by default, the SDK resets the hit bit of the FEC)

To get a range of consecutive unallocated FECs (mainly used for ECMP FECs group allocation), call `bcm_l3_egress_allocation_get(unit, flags, &egr, nof_members, &if_id);`

- `flags` – Currently redundant
- `egr.flags`:

Flags	Description
`BCM_L3_2ND_HIERARCHY`, `BCM_L3_3RD_HIERARCHY`	—

- `egr.failover_id` – Indicate whether a range of FECs must have protection
- `egr.stat_pp_profile` – If it is different than 0, the range of FECs must have statistics.
- `nof_members` – The required number of members in the range of FECs. For protection, the input variable indicates the number of protected FEC pairs.
- `if_id` – The index of the first FEC in the available range.

22.8.3 Shell Commands

- `l3 egress add`
  Create a FEC entry providing its ID and destination
- `l3 egress get`
  Receive the destination of a FEC entry
- `l3 egress find`
  Not supported
- `l3 egress update`
  Update the destination of a FEC entry

- `l3 egress delete`
  Delete an existing FEC entry by providing its ID

22.8.4 Application Reference

Example of Basic-FEC usage for IP router

- **Type:** CINT reference
- **Path:** `$SDK/src/examples/sand/cint_ip_route_basic.c`

22.9 L3 Egress Object: Protection-FEC

See Section 14.3, FEC 1:1 Unicast Protection.

22.10 L3 Egress: ECMP Object

The L3 ECMP object represents an ECMP group's FECs and group attributes that determine the method of selecting a FEC from the ECMP group, the ECMP group hierarchy stage and several other ECMP attributes which are detailed in this section.

The ECMP object ID (ECMP-ID) is located in the same range as the FEC IDs. The range is from 1 to the number of ECMPs.

Two dynamic (hashing) modes for selecting an ECMP member given a load balancing key are available:

- Multiply and divide
- Consistent

Multiply and divide dynamic mode is the default mode and the member FEC selection out of the ECMP group will be performed by adding the ECMP load balancing key modulo the ECMP group size to the first ECMP group FEC ID. This dynamic mode requires all the ECMP group members to be consecutive, which means that adding/removing members to an existing ECMP group will be to the group top or bottom.

Consistent dynamic mode provides a resilient hashing using offsets table used for reselection of the same ECMP members for the same load balancing keys (the load balancing key modulo the table size will point to an entry in the table where this entry value will be added to the ECMP base FEC ID).

Removing a member from a consistent ECMP group does not affect the selection of the remaining ECMP members by previous load balancing keys (the previous load balancing keys that selected the removed member will now select one of the remaining members with an even distribution among them). Adding a member into the group will change by one (the ECMP group size after adding the new member), and the selection of load-balancing keys changes from selecting existing members to selecting the new added member.

The following settings are possible to configure per ECMP object:

- Hierarchical-level
- Hashing mode: as mention each ECMP group can use one of two hashing modes which are:
  - Multiply and divide: The selected FEC from the group will be the base FEC ID plus an offset (with a range between 0 and the group size – 1) that is calculated using the load-balancing key and the group size.
  - Consistent hashing: In this method a consistent tables are used to hold the group FEC IDs offsets so decrementing the group size doesn't affect the remaining group members path and incrementing the group size affect only about (1/group size) of the paths.

- ECMP RPF: The ECMP has three types of RPF modes:
  - Loose RPF – The same as any unicast RPF check.
  - EM RPF – This RPF mode compares the incoming in-RIF against each of the resulting RPF lookup ECMP group members out-RIFs.
  - EM MC explicit RPF – This type of ECMP is configured as the reset of the MC RPF mode using the FEC with the matching flag and the ECMP ID which the RPF check should be done against (for more details see the IPMC section).

NOTE: If the RIF has an RPF configuration (loose or strict) and the ECMP object that was received from the RPF lookup does not have any RPF configuration (loose or EM), an RPF check based on the ECMP base FEC member will be performed.

- Statistics information – Stat-ID and Stat-profile. For more information see Chapter 15, PP Statistics Generation.

22.11 ECMP User-Defined Profile

The ECMP user-defined profile (EUDP) allows the user to create a consistent ECMP profile with custom consistent hashing table content (for example, weight support or class-based support) that can later be used to create ECMP groups.

Create the EUDP by using the `bcm_l3_egress_ecmp_user_profile_create` API. Provide the consistent table size and content, the RPF mode, and the intended ECMP hierarchy.

Any ECMP that uses this EUDP also uses its consistent hashing table and its RPF mode.

To create an ECMP group that uses an EUDP, provide the `BCM_L3_ECMP_USER_PROFILE` flag to `ecmp_group_flags` when calling the ECMP `create` API along with the EUDP profile number. Other than that requirement, the ECMP group can have any statistical configuration, a base FEC with any protection state, and a hierarchy that matches the EUDP hierarchy.

When creating an ECMP with EUDP using `bcm_l3_egress_ecmp_create`, any attributes that are provided by the EUDP (for example, RPF mode) are expected to be left unset on the ECMP create structure. Also, only the base ECMP FEC is required and not all of the FEC members as in the non EUDP case.

The consistent hashing table content of the EUDP can be updated by the user in two ways: traffic safe or space efficiency. All ECMP groups that use this EUDP will be updated with the new content. Any other EUDP parameters (such as RPF) cannot be updated after the EUDP has been created. To update the EUDP consistent table, `bcm_l3_egress_ecmp_user_profile_create` must be called with the configured `BCM_L3_REPLACE` provided to the flags field.

22.12 ECMP Group with Zero Members

A zero-member ECMP group is actually a single-member ECMP group that points to a predefined FEC. Packets that hit this ECMP group (empty group) are resolved to the destination of the internally configured member that is an FEC with a trap destination (drop FEC).

To create an empty ECMP group, first create a drop FEC that is recognized as a drop FEC by the SDK, which is a regular FEC with a trap destination that is saved per hierarchy for the use of an empty ECMP group with the same hierarchy. The drop FEC is added as a member internally. This can happen by directly calling `bcm_l3_egress_ecmp_create` with 0 interfaces or by deleting the last member of an ECMP group using `bcm_l3_egress_ecmp_delete`.

22.12.1 ECMP Tunnel Priority

The BCM88830 introduces the ECMP tunnel priority (TP) feature, which is an extended version of the ECMP *consistent* dynamic mode.

The TP mode allows using several consistent tables for a single ECMP group when the incoming packet's tunnel priority value selects between the consistent tables. This way, each set of tunnel priority values can lead to a different set of forwarding paths.

The tunnel priority is a 3-bit value taken from the IP ECN field (2 bits) and the LSB bit of the IP DSCP field, but it can be updated by a 5-bit value using the PMF.

The TP requires the same basic *consistent* dynamic mode configurations that determine the table sizes in the same way (all the ECMP TP tables are the same size). The only exception for TP is when creating the ECMP group, the base FEC should be the only group member to be provided to the `bcm_l3_egress_ecmp_create` API, along with a tunnel priority mode and the tunnel-priority map profile. The ECMP members will be added separately to each table using a dedicated API, `bcm_l3_egress_ecmp_tunnel_priority_set`.

The ECMP tunnel-priority mode determines the number of tables used for the TP (two, four, or eight) or if the TP mode is disabled.

The ECMP tunnel-priority map profile is one of four available profiles (each ECMP hierarchy has four profiles) that map the packet tunnel-priority value into the matching ECMP TP table.

ECMP TP can also be achieved in the BCM88830 and other DNX family devices by changing the FEC or ECMP destination in the PMF according to *tunnel priority*, which can be derived from any internal or packet QoS attribute, such as TC, NWK-QOS, MPLS.EXP, or IP.DSCP. When using a single FEC hierarchy, this can be achieved by changing the FEC destination of the packet using `bcmFieldActionForward`. For more information on PMF actions, see Chapter 11, Field Processor. To use two or three FEC hierarchies, use the following steps.

1. Create ECMP configured as multiply and divide. The size of the ECMP must be a power of 2.
2. According to the tunnel priority values, modify the log2n of the ECMP size MSB bits of the LB-key using the PMF hash `bcm_field_context_hash_create()`.
   For more information on PMF hashing, see Section 11.29, Hashing Parameters.

22.12.2 SOC Properties

None

22.12.3 Configuration Flow

To create an ECMP object call: `bcm_l3_egress_ecmp_create(unit, ecmp, intf_count, intf_array)`

- `ecmp.flags`:

Flags	Description
`BCM_L3_WITH_ID`	If omitted, the ECMP interface ID will be assigned by the SDK, otherwise the value of the `ecmp.ecmp_intf` will be used
`BCM_L3_REPLACE`	Replace the ECMP group with the given ID with the provided ECMP group (`BCM_L3_WITH_ID` must be present when used)
`BCM_L3_2ND_HIERARCHY`	—

Flags	Description
BCM_L3_3RD_HIERARCHY	—

- ecmp.ecmp_group_flags

Flags	Description
BCM_L3_ECMP_USER_PROFILE	Use a user-defined profile for this ECMP group. If this flag is set then: ■ The rpf_mode should be set to default. ■ The tunnel priority mode should be disabled. ■ The max_paths should be set to 0. As all of the above are derived from the EUDP itself.
BCM_L3_ECMP_MEDIUM_TABLE	—
BCM_L3_ECMP_LARGE_TABLE	—
BCM_L3_ECMP_EXTENDED	—

- ecmp.ecmp_intf – If the BCM_L3_WITH_ID flag is present use this value as the ECMP interface ID.
- ecmp.max_paths – The maximal number of members that the ECMP group can hold. Relevant for consistent hashing .
- ecmp.dynamic_mode:
  - BCM_L3_ECMP_DYNAMIC_MODE_DISABLED (default) – Use multiply and divide as the hashing method.
  - BCM_L3_ECMP_DYNAMIC_MODE_RESILIENT – Use consistent hashing as the hashing method.
- ecmp.stat_id – Object statistics ID (valid only when stat_pp_profile is not 0)
- ecmp.stat_pp_profile – Stat-PP profile

  For more information regarding the creation of statistics interfaces and counting, see Chapter 15, PP Statistics Generation.
- ecmp.tunnel_priority.mode – The tunnel priority (TP) mode:

Mode	Description
bcmL3EcmpTunnelPriorityModeDisable	The TP mode is disabled (Default)
bcmL3EcmpTunnelPriorityModeTwoPriorities	Use two priorities tables for the ECMP TP.
bcmL3EcmpTunnelPriorityModeFourPriorities	Use four priorities tables for the ECMP TP.
bcmL3EcmpTunnelPriorityModeEightPriorities	Use eight priorities tables for the ECMP TP.

NOTE: The BCM88690, BCM88480, and BCM88000 devices do not support the tunnel priority feature. For these devices, the only valid value is disable.

- ecmp.tunnel_priority.map_profile – The tunnel priority mapping profile that maps between the tunnel priority values and the consistent tables.
- ecmp.tunnel_priority.index – Set to 0 for the create API. For the get, add, and delete APIs, this index indicates the serial number of the requested TP table to get, add, or delete.
- ecmp.rpf_mode – Unicast RPF modes. The following three modes are supported bcmL3EcmpUrpfInterfaceDefault, bcmL3EcmpUrpfLoose, bcmL3EcmpUrpfStrictEm.
- intf_count – The size of the intf_array. The value can be 0 when setting a drop FEC.
- intf_array – Array of FEC IDs. It can be empty when setting a drop FEC.
- ecmp.user_profile – The user-defined profile number (valid only when the BCM_L3_ECMP_USER_PROFILE flag is indicated).

To create a user-defined ECMP profile (EUDP), call the following API:

`bcm_l3_egress_ecmp_user_profile_create(unit, ecmp, intf_count, intf_array)`

- `ecmp.flags`

Flags	Description
`BCM_L3_WITH_ID`	If omitted, the EUDP ID is assigned by the SDK; otherwise, the value of `ecmp.user_profile` is used.
`BCM_L3_REPLACE`	Replace the EUDP with the given ID content. (`BCM_L3_WITH_ID` must be present when used.) Also, only the members in the consistent table can be replaced and no other EUDP parameters can be specified.
`BCM_L3_2ND_HIERARCHY`	—
`BCM_L3_3RD_HIERARCHY`	—

- `ecmp.ecmp_group_flags`

Flags	Description
`BCM_L3_ECMP_MEDIUM_TABLE`	—
`BCM_L3_ECMP_LARGE_TABLE`	—
`BCM_L3_ECMP_RESILIENT_REPLACE`	—

NOTE: If no group flag is indicated, a small table with 128 rows and up to 16 members is selected.

- `ecmp.dynamic_mode:`
  - `BCM_L3_ECMP_DYNAMIC_MODE_RESILIENT` – Use consistent hashing as the hashing method (must be indicated).
  - `ecmp.rpf_mode` - Unicast RPF mode. The following three modes are supported:
    - `bcmL3EcmpUrpfInterfaceDefault`
    - `bcmL3EcmpUrpfLoose`
    - `bcmL3EcmpUrpfStrictEm`
  - `intf_count` – The size of `intf_array` should have the size of the selected consistent hashing table.
  - `intf_array` – Array of member offsets. The array should contain offsets in a range according to the selected consistent hashing table.

To support an ECMP group with zero members, it is required to configure a FEC to be a Drop-FEC:

- `bcm_switch_control_indexed_set(unit, fec_drop_key, fec_drop_info)`
  - `fec_drop_key.type = bcmSwitchEcmpEmptyEgressId` to indicate configuring the drop FEC destination for the use of ECMP.
  - `fec_drop_key.index` – Hierarchy level-1.
  - `fec_drop_info.value` – Value of the FEC index with a trap drop destination.

If tunnel priority mapping is not disabled, the following steps are required *before* creating an ECMP object:

1. Create a tunnel-priority mapping profile by calling `bcm_l3_ecmp_tunnel_priority_map_create(unit, map_info)`
   - `map_info.l3_flags:`

Flags	Description
`BCM_L3_WITH_ID`	If omitted, an available map profile ID is allocated. Otherwise, the `map_info.map_profile` value is used for the allocation.
`BCM_L3_2ND_HIERARCHY`	—
`BCM_L3_3RD_HIERARCHY`	—

   - `map_info.map_profile` – If the `BCM_L3_WITH_ID` flag is used, this field value is used as the profile ID. Otherwise, the allocated profile is returned using this field. For the update API, it indicates the allocated profile.
2. To update the mapping between a tunnel priority (received from the packet and/or PMF) to an ECMP TP table, call `bcm_l3_egress_ecmp_tunnel_priority_set(unit, map_info)`.
   - `map_info.l3_flags` – The same as the create API (described in the previous step) except `BCM_L3_WITH_ID`.
   - `map_info.map_profile` – The profile ID value from the creation API.
   - `map_info.tunnel_priority` – The tunnel priority value to map to the consistent table placed on the `map_info.index` index.
   - `map_info.index` – The serial index of the consistent table the `map_info.tunnel_priority` value points to.

22.12.4 Shell Commands

- `l3 ecmp create`
  Create an ECMP group using ID, size and an array of members.
- `l3 ecmp add`
  Add a member to an already existing ECMP group (or one of the ECMP TP-consistent tables if TP is enabled).
- `l3 ecmp delete`
  Delete a member from an already existing ECMP group (or one of the ECMP TP-consistent tables if TP is enabled).
- `l3 ecmp destroy`
  Destroy an existing ECMP group providing its ID.
- `l3 ecmp dump`
  Print all existing ECMP groups and their data.
- `l3 ecmp tunnel priority set`
  Set a single ECMP TP table members
- `l3 ecmp tunnel priority map create`
  Create a map profile between the packet TP value to the ECMP TP table
- `l3 ecmp tunnel priority map destroy`
  Destroy an existing TP mapping profile.
- `l3 ecmp tunnel priority map set`
  Set a single mapping entry between a TP value to a ECMP TP table
- `l3 ecmp tunnel priority map get`
  Get a single mapping entry between a TP value to a ECMP TP table

22.12.5 Application Reference

Example of Basic-ECMP usage

- **Type:** CINT reference
- **Path:** `$SDK/src/examples/sand/cint_ecmp_basic.c`

Example of ECMP tunnel priority

- **Type**: CINT reference
- **Path:**
  - `$SDK/src/examples/dnx/cint_cbts.c` (BCM88690, BCM88800, BCM88480)
  - `$SDK/src/examples/sand/cint_ecmp_basic.c` (BCM88830)

Example of ECMP user profile configuration

- **Type**: CINT reference
- **Path**: `$SDK/src/examples/dnx/l3/cint_dnx_ecmp_eudp_example.c`

Example of ECMP RPF configuration

- **Type**: CINT reference
- **Path**: `$SDK/src/examples/dnx/l3/cint_ip_ecmp_rpf_examples.c`

Example of ECMP group configuration with zero members:

- **Type**: CINT reference
- **Path**: `$SDK/src/examples/dnx/l3/cint_ecmp_basic.c`

22.13 Adding Route Entries

The route entries are used for either RPF or forwarding. Unlike the host entries the only supported destination for a UC route entry is a FEC.

It is, however, also possible to load *raw* entries (see the flag information in Section 22.13.5, Configuration Flow). In that case, the *destination* is loaded as is, without any encoding and any 20-bits value is acceptable. This is mainly used for *destinations* that are an input to an ACL to carry out some other action than the standard.

The route entry key is composed of VRF and destination IP (IPv4 or IPv6) for a private search, and only destination IP for a public search. A VRF=0 will indicate a public entry if such database exists and for a single private database, the VRF=0 will be used as any other VRF value.

22.13.1 Traverse Route

The traversal function allows the user to go over all entries and exercise a particular action using the callback function. There is a limitation regarding the actions which can be taken on LPM entries, however, they cannot be deleted using the callback function. `*_delete_all` and `*_remove_all` are designated APIs for this purpose.

22.13.2 Default Route

If the IP address is fully masked the entry is considered to be a default route entry which could be either public or private depending on the VRF value. This is similar to the non default route entries.

NOTE: When the `bcmSwitchL3UrpfDefaultRoute` switch option (using the `bcm_switch_control_set` API) is set to 1, the default routes are included in the UC loose and strict RPF lookups.

22.13.3 Route Hit Bits

Hit bits indicate which entry from the route table was hit (matched) on the last packet forwarding. For route tables, this functionality is not enabled by default. It is possible to enable or disable this functionality per Logical-table.

22.13.4 SOC Properties

None

22.13.5 Configuration Flow

Global configuration:

1. Configure the default route behavior for unicast RPF lookups using the `bcmSwitchL3UrpfDefaultRoute` switch option of the `bcm_switch_control_set` API.
2. Enable or disable the hit bit for any routing table using the API `bcm_switch_control_indexed_set` with type `bcmSwitchL3LpmHitbitEnable`. The following values exist:
   - `bcmL3LpmHitbitTableIPv4UC`
   - `bcmL3LpmHitbitTableIPv6UC`
   - `bcmL3LpmHitbitTableIPv4MC`
   - `bcmL3LpmHitbitTableIPv6MC`

To add a route entry, call `bcm_l3_route_add(unit, &route_info)`

- `route_info.flags:`

Flags	Description
`BCM_L3_IP6`	Add an IPv6 route entry (IPv4 entry if omitted).
`BCM_L3_REPLACE`	Replace current route entry with the provided route entry. The flag is not mandatory to update an entry, but if it has been provided and entry does not exist, an error is returned

- `route_info.flags2`:

Flags	Description
`BCM_L3_FLAGS2_RAW_ENTRY`	If present, then raw data is loaded as is to KAPS result, without encoding, using the value in `route_info.l3a_intf`.
`BCM_L3_FLAGS2_SCALE_ROUTE`	Add a route entry to the private KAPS DB.

- `route_info.l3a_vrf` –the VRF value that will be part of the private route entry key (this value cannot be partly or fully be masked), if this field is equal to zero and a public database exists it will indicate the use of such.
- route_info.l3a_subnet – The IPv4 IP destination subnet address that will be part of the route key (not relevant if the `BCM_L3_IP6` flag present).
- `route_info.l3a_ip_mask` – The IPv4 subnet mask (not relevant if the `BCM_L3_IP6` flag present).
- `route_info.l3a_ip6_addr` – The IPv6 IP destination subnet address that will be part of the route key (not relevant if the `BCM_L3_IP6` flag omitted)
- `route_info.l3a_ip6_mask` the IPv6 subnet mask (not relevant if the `BCM_L3_IP6` flag omitted).
- `route_info.l3a_intf` – Holds the destination FEC ID. See remark, concerning `BCM_L3_FLAGS2_RAW_ENTRY`, above.

22.13.6 Shell Commands

- `l3 route add`
  Create a route entry providing its IP, mask, VRF and destination
- `l3 route update`
  Update the destination of a route entry
- `l3 route get`
  Receive the destination of a route entry. Note that if the `BCM_L3_FLAGS2_RAW_ENTRY` flag is set then entry is retrieved as is, without encoding. Otherwise, entry is retrieved with encoding unless validity discovers that it was added as 'raw'. In that case, the `BCM_L3_FLAGS2_RAW_ENTRY` flags is set, on output, and entry is retrieved as is, without encoding.
- `l3 route delete`
  Delete a route entry providing its IP, mask and VRF.

22.13.7 Application Reference

Example of basic-route API usage:

- **Type**: CINT reference
- **Path:** `$SDK/src/examples/sand/cint_ip_route_basic.c`

Example of basic-route API usage with a *raw* destination:

- **Type:** CINT reference
- **Path:** `$SDK/src/examples/sand/cint_ip_route_basic.c`

22.14 Adding IPMC Entries

This subsection explains the IPMC flows (both IPv4 MC flows and IPv6 MC flows) and the required configuration for adding IPMC entries.

22.14.1 IPv4 MC Flow

In an IPv4 compatible multicast destination address the following statements are true:

- DA[47:24]= {24'h01_00_5E, 1'b0}
- DIP[22:0]=DA[22:0]
- DIP[31:28]={4'hE}

This section details the three IPv4 multicast flows resulting in the L2 split to to 4V4MC and bridge forwarding.

The selected flow is determined by the incoming RIF properties, including routing enablers, the L2 IPMC forwarding type, and the MAC table mode. For more information about RIF properties, see Section 22.6, ETH-RIF (L3 VSI Interface) and VRF Definition.

The default value of the L2 IPMC forwarding type can be determined as follows:

- For IPv4, call `bcm_switch_control_set(unit, bcmSwitchL3McastL2, config)`
  `config` – Default value for VSI forwarding type: v4Bridge (0) or IPv4MC (1).
- For IPv6, call `bcm_switch_control_set(unit, bcmSwitchL3IP6McastL2, config)`
  `config` – Default value for VSI forwarding type: v6Bridge (0) or IPv6MC (1).

NOTE: The default L2 IPMC forwarding will be overwritten when the `bcm_l3_intf_create` API is called.

The following table describes which RIF flags are used for each flow.

Flow		Required Flags
L3		None
L2	bridge forwarding	BCM_L3_INGRESS_ROUTE_DISABLE_IP4_MCAST
	v4MC	BCM_L3_INGRESS_ROUTE_DISABLE_IP4_MCAST BCM_L3_INGRESS_L3_MCAST_L2

22.14.1.1 L3 Flow

All the L3 flows keys contains an InLIF as part of the forwarding key, the InLIF can be either the incoming ETH-RIF or a global L3 Tunnel for tunnel termination (for example, MPLS/IP tunnels).

If the incoming ETH-RIF/L3-Tunnel is IPv4-IPMC-enable (the default configuration of the RIF), the L3 flow of the IP MC is selected.

Table 84: IPv4 MC L3 Forwarding Lookup

Number	Key Fields	Database Type
1	<VRF, G , InLIF>	LEM
2	<VRF, G, SIP, InLIF>	KAPS (private lookup)
3	<0, G, SIP, InLIF>	KAPS (public lookup)
4	<VRF, G, SIP, InLIF>	TCAM

The RPF lookups of the L3 flows are listed in the following table. RPF lookups are performed in the unicast tables.

Table 85: IPv4 MC L3 RPF Lookups

Number	Key Fields	Database Type
1	<VRF,SIP>	LEM
2	<VRF,SIP>	KAPS (private lookup)
3	<0,SIP>	KAPS (public lookup)

22.14.1.2 L2 Flow

If the in-RIF is IPMC-disable the L2 flow is selected and one of the following flows is used

- **v4MC**
  If the RIF was created with the BCM_L3_INGRESS_L3_MCAST_L2 flag this flow is selected and the following forwarding lookups are performed.

Table 86: IPv4 MC Forwarding Lookups

Number	Key Fields	Database Type
1	<FID,G>	LEM
2	<FID,G,SIP>	KAPS

- **Bridge forwarding**
  This is the default L2 flow (BCM_L3_INGRESS_L3_MCAST_L2 flag was omitted on RIF creation). The forwarding lookups is shown in the following table.

Table 87: Forwarding Lookups for Bridge Forwarding

Number	Key Fields	Database Type
1	<FID,DA>	LEM

22.14.2 IPv6 MC Flow

In an IPv6 compatible multicast destination address the following statements are true:

- DA[47:32]= {16'h33_33}
- DIP[31:0]=DA[31:0]
- DIP[127:120]={8'hFF}

The following table describes which RIF flags are to be used for each one of the flows

Table 88: IPv6 MC Flows Selection Flags

Flow	Function	Required Flags
L3	—	None
L2	Bridge Forwarding	`BCM_L3_INGRESS_ROUTE_DISABLE_IP6_MCAST`
	v6MC	`BCM_L3_INGRESS_ROUTE_DISABLE_IP6_MCAST` `BCM_L3_INGRESS_IP6_L3_MCAST_L2`

22.14.2.1 L2 Flow

If the in-RIF is IPMC-disable, the L2 flow is selected, and one of the following flows is used:

- **v6MC**
  If the RIF was created with the `BCM_L3_INGRESS_L3_IP6_MCAST_L2` flag, this flow is selected, and the forwarding lookup shown in the following table is performed. For more information, see Section 22.15, IPv6 MC Cascaded.

Forwarding Lookups for IPv6 MC		
No.	**Key Fields**	**Database Type**
1	<FID,G,SIP>	KAPS cascaded

- **Bridge forwarding**
  This is the default L2 flow (the `BCM_L3_INGRESS_L3_IP6_MCAST_L2` flag was omitted on RIF creation). The forwarding lookup is shown in the following table.

Forwarding Lookups for Bridge Forwarding		
No.	**Key Fields**	**Database Type**
1	<FID,DA>	LEM

22.14.2.2 L3 Flow

The EM L3 flows keys contain an InLIF as part of the forwarding key, the InLIF can be either the incoming ETH-RIF or a global L3 Tunnel for tunnel termination (for example, MPLS/IP tunnels). The KAPS L3 flow key contains a compressed source and interface value (C_SI) that is part of the cascaded lookup (see Section 22.15, IPv6 MC Cascaded).

If the incoming ETH-RIF/L3-Tunnel is IPv6-IPMC-enable (the default configuration of the RIF) ,the L3 flow of the IPv6 MC is selected. The forwarding lookups are listed in the following table.

Table 89: IPv6 MC L3 Forwarding Lookups

No.	Key Fields	Database Type
1	<VRF,G,In-LIF>	LEM
2	<VRF,G,SIP,In-LIF>	KAPS (private cascaded lookup)
3	<VRF,G,SIP,In-LIF>	TCAM

The RPF lookups are identical to those of the IPv4 MC L3 flows.

22.14.3 Multicast Group and Replications

The multicast group object is composed of an MC group ID and replications. The MC group ID serves as a destination in the system for flows that are replicated according to the pointed group.

The MC group can be an ingress MC group or an egress MC group or both. Selection of the replication point (ingress/egress) for each replication determined according to the replication configuration on replication creation.

For more information about multicast APIs, see the Multicast section in the *Traffic Manager* programming guide.

The replications determine the number of instances that are replicated from the incoming packet and the destination of each one.

Each replication has a single field for an encapsulation object and a destination.

If more encapsulation objects are needed, the extension encapsulation table is used for up to three encapsulation objects (see `bcm_multicast_replication_t`).

22.14.4 Adding an IP L3 Multicast Entry

The IP L3 multicast flows (see in the IPv4 and IPv6 IPMC diagram section L3 flows) includes both host and route lookups for forwarding and for RPF checks.

All the IPMC forwarding entries are added using the same API (`bcm_ipmc_add`). The IP version is determined using the flag `BCM_IPMC_IP6`. The database type of entry is determined according to the key field values and key field masks as described in the following table.

When working with a single KAPS DB (KAPS2 capacity is 0), LEM and KAPS cannot have a masked VRF value. Valid entries include the following:

- KAPS entry with literal value of VRF = 0 (saved in KAPS1);
- KAPS entry with VRF value different than 0 (saved in KAPS1);
- LEM entry with literal value VRF = 0;
- LEM entry with VRF value different than 0.

Table 90: IPv4 MC L3 Flow Database Selection

Database Type	Selection Conditions
Host (LEM)	InLIF is not masked MC group IP is not masked VRF > 0 or VRF = 0 when KAPS2 capacity is 0 SIP is fully masked
TCAM	`BCM_IPMC_TCAM` flag is provided
LPM (KAPS)	Not host or TCAM The key has an LPM mask

Table 91: IPv6 MC L3 Flow Database Selection

Database Type	Selection Conditions
Host (LEM)	InLIF is not masked MC group IP is not masked VRF > 0 or VRF = 0 when KAPS2 capacity is 0 SIP is fully masked
TCAM	`BCM_IPMC_TCAM` flag is provided
LPM (KAPS)	Not host or TCAM The key has an LPM mask VRF > 0 or VRF = 0 when KAPS2 capacity is 0.

NOTE:

- Fully masked – The field mask is equal to zero, and all the field bits are under a *do not care* state
- LPM mask – A consecutive mask that start from the LSB.
- To add the source IP address to the lookup key for LPM IPv6 MC tables, a cascaded lookup was introduced. For cascaded IPv6 MC lookup information, see Section 22.15, IPv6 MC Cascaded.

If no database matched the entry parameters, an error message is returned.

Adding an IP multicast RPF check entry (host and route) is done using the unicast APIs `bcm_l3_host_add` and `bcm_l3_host_route`. The L3 IPMC KAPS entries (RPF and forwarding) are public when the VRF value is 0 and a public database is available, or private in any other case.

22.14.5 Adding a v4MC Bridged Entry

The v4MC L2 flow has two entry types:

- A host entry placed in the LEM with a key that is composed from an FID and a group.
- An LPM route entry placed in the KAPS with a key that is composed from an FID, group and a SIP.

The selection between the two entry types is determined according to the keys and masking values.

IPv4 v4MC Bridge Database Selection	
Database Type	**Selection Conditions**
Host (LEM)	Group is not masked SIP fully masked
LPM (KAPS)	The key have an LPM mask

22.14.6 Adding a v6MC Bridged Entry

The v6MC L2 flow has one entry type:

An LPM bridge entry placed in the KAPS with a key that is composed from an FID, IPv6 group and an IPv6 SIP. For cascaded IPV6 MC lookup, see Section 22.15, IPv6 MC Cascaded.

IPv6 v6MC Bridge Database Selection	
Database Type	**Selection Conditions**
LPM (KAPS)	The key has an LPM mask

22.14.7 SOC Properties

None

22.14.8 Configuration Flow

22.14.8.1 Adding an IP L3 Multicast Entry

To add an L3 IPMC entry, call: `bcm_ipmc_add(unit, &ipmc_info)`

- `ipmc_info.flags:` supported flags:

Supported Flags	**Description**
`BCM_IPMC_IP6`	Add an IPv6 host entry (IPv4 entry if omitted).
`BCM_IPMC_REPLACE`	Replace current host entry with the provided host
`BCM_IPMC_RAW_ENTRY`	The value of `l3a_intf` will loaded as raw data into the LPM table. Supported only for LPM table and not, for example TCAM database.
`BCM_IPMC_HIT`	Indicates whether the entry has been hit when calling `bcm_ipmc_find`.

- `ipmc_info.s_ip_addr` – To specify an IPv4 SIP address, compressed_SIP (for IPv6), or the compressed_SI (for IPv6)
- `ipmc_info.mc_ip_addr` – To specify an IPv4 MC group.
- `ipmc_info.mc_ip6_addr` – To specify an IPv6 MC group.
- `ipmc_info.s_ip_mask` – To specify an IPv4 SIP mask, compressed_SIP (for IPv6), or the compressed_SI (for IPv6) mask.
- `ipmc_info.mc_ip_mask` – To specify an IPv4 group mask.
- `ipmc_info.mc_ip6_mask` – To specify an IPv6 group mask.
- `ipmc_info.vid` – To specify the InLIF.
- `ipmc_info.vrf` – To specify the VRF value.
- `ipmc_info.group` – To specify the MC group index.
- `ipmc_info.l3a_intf` – To specify a, FEC/ECMP ID. This option is typically used when an MC RPF is required (for more information, see Section 22.16, Multicast RPF) or when creating default entries that are FEC attributes.
- `ipmc_info.ing_intf` – To specify LIF interfaces that are not ETH-RIF.
- `ipmc_info.priority` – Relevant only with the `BCM_IPMC_TCAM` flag. Indicates the priority of the entry in the TCAM table, 0 being the highest priority.
- `ipmc_info.stat_id` – Object statistics ID
- `ipmc_info.stat_pp_profile` – PP statistics ID and profile

For more information regarding the creation of statistics interfaces and counting, see Chapter 15, PP Statistics Generation

NOTE: The `group`, `l3a_intf` and the `port_tgid` fields are mutually exclusive.

22.14.8.2 Adding a v4MC Entry

To add an L2 v4MC entry call: `bcm_ipmc_add(unit, ipmc_info)`

- `ipmc_info.flags:`

Supported Flags	Description
`BCM_IPMC_L2`	Add an L2 v4MC host entry (this flag is mandatory for adding a v4MC entry).
`BCM_IPMC_REPLACE`	Replace current host entry with the provided host

- `ipmc_info.mc_ip_addr` – To specify the IPv4 MC group.
- `ipmc_info.s_ip_addr` – To specify the IPv4 SIP.
- `ipmc_info.mc_ip_mask` – To specify the IPv4 MC group mask.
- `ipmc_info.s_ip_mask` – To specify the IPv4 SIP mask.
- `ipmc_info.vid` – To specify the FID value.
- `ipmc_info.group` – To specify the MC group index (ID that is created by Multicast APIs).

22.14.8.3 Adding a v6MC Bridged Entry

To add a bridge forwarding entry, see Section 21.14, L2 MACT Forwarding.

To add an L2 v6MC entry, call `bcm_ipmc_add(unit, ipmc_info)`.

- `ipmc_info.flags`:

Supported Flags	Description
BCM_IPMC_L2	Add an L2 MC entry (mandatory)

Supported Flags	Description
BCM_IPMC_IP6	Add an IPv6 entry (mandatory)
BCM_IPMC_REPLACE	Replace current host entry with the provided host

- `ipmc_info.mc_ip6_addr` – To specify the IPv6 MC group.
- `ipmc_info.s_ip_addr` – To specify the Compressed SIP.
- `ipmc_info.mc_ip6_mask` – To specify the IPv6 MC group mask.
- `ipmc_info.s_ip_mask` – To specify the Compressed SIP mask.
- `ipmc_info.vid` – To specify the FID value.
- `ipmc_info.group` – To specify the MC group index (ID that is created by Multicast APIs).

22.14.9 Shell Commands

None

22.14.10 Application Reference

Example of Basic-L3-IP-MC API usage

- **Type:** CINT reference
- **Path:** `$SDK/src/examples/sand/cint_ipmc_route_basic.c`

Example of Basic-L2-v4MC API usage

- **Type:** CINT reference
- **Path:** `$SDK/src/examples/sand/cint_sand_ip_compatible_mc.c`

Example of Basic-L2-MC-bridge-forwarding API usage

- **Type:** CINT reference
- **Path:** `$SDK/src/examples/sand/cint_sand_ip_compatible_mc.c`

Example of Basic-L2-v6MC API usage

- **Type**: CINT reference
- **Path**: `$SDK/src/examples/sand/cint_ipmc_route_examples.c`

22.15 IPv6 MC Cascaded

The *cascaded lookup* feature adds the source IP address to the lookup key for LPM tables used in IP multicast IPv6 forwarding.

The cascaded lookup is carried out in two steps in the FWD stage. The first-stage lookup is carried out as follows:

- For an IPv6 MC route entry – The source IP address of the packet and the RIF value are used to find a 16-bit user-defined value that represents the source and interface (compressed_SI) field.
- For an IPv6 MC bridge entry – The source IP address of the packet is used to find a 16-bit user-defined value that represents the source IP value (compressed_SIP).

The second stage uses the compressed_SIP or the compressed_SI value as part of the key for the LPM lookup as follows:

- For an IPv6 MC route entry – The lookup is carried out using the VRF, IPMC Group, and the Compressed_SI value.
- For the IPv6 MC bridge entry – The lookup is carried out using the forwarding ID, IPMC group, and the compressed_SIP.

22.15.1 Configuration Flow

To configure a cascaded entry, the `bcm_ipmc_config_add` API is introduced. This API creates a compressed_SIP or a compressed_SI entry.

- `ipmc_config.flags`:

Supported Flags	Description
`BCM_IPMC_REPLACE`	Replaces the current entry with the provided new destination.
`BCM_IPMC_L2`	Indicates a compressed_SIP entry. If a flag is not provided, a compressed_SI entry will be created.

- Key fields:
  - `ipmc_config.s_ip6_addr` – To specify an IPv6 SIP.
  - `ipmc_config.s_ip6_mask` – To specify an IPv6 SIP mask.
  - `ipmc_config.vid` – To specify the InLIF (used for Compresssed_SI entry)
- Result:
  - `ipmc_config.s_ip_addr` – Specify the compressed value.

22.15.2 Application Reference

Example of Cascaded L3-MC API usage

- **Type:** CINT reference
- **Path:** `$SDK/src/examples/sand/cint_ipmc_route_basic.c`

22.16 Multicast RPF

The MC RPF information resides on the forwarding FEC resulted from the MC forwarding lookup.

To set an MC RPF, an FEC that contains the MC RPF type is used as part of the MC entry result.

NOTE: If the RPF lookup of a MC packet returns an ECMP and the MC RPF type is `SIP_BASED`, the MC RPF check is disregarded and `SIP_BASED` RPF trap is not raised.

The available MC RPF types are:

- `BCM_L3_MC_RPF_SIP_BASE` – Verifies that the outgoing RIF of the RPF lookup (using the SIP from the packet) is the same as the incoming RIF of the MC packet.
- `BCM_L3_MC_RPF_EXPLICIT` – Verifies that the MC incoming RIF is the same as the RIF that was found from the forwarding FEC.
- `BCM_L3_MC_RPF_EXPLICIT_ECMP` – Verifies that the incoming RIF is one of an ECMP group outgoing RIFs.
- `BCM_L3_MC_NO_RPF` – If the incoming RIF holds any UC RPF settings, a UC RPF check is performed over the MC packet, unless a MC RPF check exists. This flag is provided in case MC/UC RPF checks are not required but the incoming RIF has a UC RPF configuration.

If one of the explicit RPF modes is used the `bcm_l3_egress_t.intf` field that is received by the `bcm_l3_egress_create` API holds one of the following:

- The expected in-RIF for `BCM_L3_MC_RPF_EXPLICIT`
- The ECMP ID for `BCM_L3_MC_RPF_EXPLICIT_ECMP`.

NOTE: Setting a MC explicit ECMP RPF requires additional configuration in the ACL block (see `cint_ip_ecmp_rpf_examples.c`).

22.16.1 Configuration Flow

A default route hit may affect SIP-based MC RPF according to the following configuration:

1. Call `bcm_switch_control_set(unit, type, arg)`
   - `type` – `bcmSwitchL3McRpfDefaultRoute`
   - `arg` – A value of 1 means the default route affects SIP-based RPF. A value of 0 means the default route does not affect SIP-based RPF.
2. Create FEC MC-RPF by calling `bcm_l3_egress_create` with the fields and flags explained previously.

After creation, the FEC ID is placed on the `l3a_intf` field of the `bcm_ipmc_addr_t` structure that is used by the `bcm_ipmc_add` API. See more information in Section 22.14, Adding IPMC Entries.

22.16.2 Application Reference

Example of Basic-L3-MC-RPF API usage

- **Type:** CINT reference
- **Path:** `$SDK/src/examples/sand/cint_ip_route_rpf_basic.c`

22.17 Adding Default IPMC Entries

A default IPMC entry is an entry with the lowest priority in a KAPS/KBP table. This type of entry can be created in the following two ways:

- All maskable key fields are masked (G, S, RIF) in the `bcm_ipmc_addr_t` structure;
- The `BCM_IPMC_DEFAULT_ENTRY` flag is provided to the `flags` field of the `bcm_ipmc_addr_t` structure.

When creating a default MC entry, a default FEC must be used as the entry destination.

22.17.1 Application Reference

Example of adding a default IPMC entry

- **Type**: CINT reference
- **Path**: `$SDK/src/examples/sand/cint_ipmc_route_priorities.c`
  `$SDK/src/examples/sand/cint_ipmc_route_basic.c`

22.18 Bridge Fallback

If an unsuccessful IPv4- or IPv6-compatible MC forwarding lookup occurs, the forwarding layer reverts back to L2 and the action of the *unknown address* configuration is performed.

For more information about the unknown address configuration, see Section 21.4, Port Bridging Properties.

NOTE: For the BCM88690, BCM88800, and BCM88480 devices, it is required to add a Field-Processor application of two-pass flow in addition to enabling the flag.

Bridge fallback can be enabled per routing interface.

22.18.1 Configuration Flow

On an existing RIF, call `bcm_l3_ingress_create` with the `BCM_L3_INGRESS_IPMC_BRIDGE_FALLBACK` flag.

For the BCM88690, BCM88800, and BCM88480 devices, it is required to add a Field-Processor application. For a field processor example configuration, refer to `cint_field_bridge_fallback_main` and `cint_field_bridge_fallback_entry_add` in `cint_field_bridge_fallback.c`.

22.18.2 Application Reference

Example of Bridge fallback

- **Type:** CINT reference
- **Path:** `$SDK/src/examples/sand/cint_ipmc_route_examples.c`
-

22.19 L3 Filters

A set of ingress and egress filters are enabled by default to filter incorrect packets going through the pipeline. The below table summarizes all ingress traps that are enabled by default and the action taken for the packets, qualifying on the traps:

L3 Filter Traps Configured on init	Description	Default Action
`bcmRxTrapMcExplicitRpfFail`	Filter Packets with MC Explicit RPF failure	Drop
`bcmRxTrapMcUseSipRpfFail`	Filter Packets MC SIP RPF failure	Drop
`bcmRxTrapUcLooseRpfFail`	Filter Packets with loose RPF failure	Drop
`bcmRxTrapUcStrictRpfFail`	Filter Packets with strict RPF failure	Drop
`bcmRxTrapUnknownDest`	Filter IP packets for which no forwarding destination has been found.	Drop

The following table summarizes all egress traps that are enabled by default and the action taken for the packets, qualifying on the traps:

L3 Filter Traps Configured on init	Description	Default Action
`bcmRxTrapEgIpv4Ttl0`	Filter IP packets with TTL=0	Drop
`bcmRxTrapEgIpv4Ttl1`	Filter IP packets with TTL=1	Drop
`bcmRxTrapEgIpv4Error`	Filter Packets with IPv4 Error indication	Drop

22.19.1 SOC Properties

None

22.19.2 Configuration Flow

Configure traps to different behavior by using the RX Trap APIs. For more information, see Chapter 12, Traps.

To configure packet filtering with a particular trap if no valid forwarding destination has been found, take the following steps:

1. Create a user-defined trap (see Chapter 12, Traps) and configure it with the required destination port.
2. Call the following API with the received trap ID:

```
bcm_switch_control_set(unit, bcmSwitchForwardLookupNotFoundTrap, trap_id);
```

NOTE: IPv6 MC forwarding is not supported with `bcmSwitchForwardLookupNotFoundTrap`.

22.19.3 Shell Commands

None

22.19.4 Application Reference

None

22.20 Performance Improvement

There are several ways to improve performance for L3 APIs by disabling some functionalities. These suggestions are not recommended for API calls that might return an error.

22.20.1 Configuration Flow

To disable one of the functionalities with the aim of reducing the runtime of the APIs, the following API can be used:

```
bcm_switch_indexed_control_set(int unit, bcm_switch_control_key_t key,
bcm_switch_control_info_t info);
```

Where `info` holds the value for enabling or disabling a functionality:

- `info.value` = 0: Disable functionality
- `info.value` = 1: Enable functionality

The `key` variable is configured as follows:

Disabling error recovery

`key.type` = `bcmSwitchModuleErrorRecoveryEnable`;

Disabling verification functions

`key.type` = `bcmSwitchModuleVerifyEnable`;

`key.index` can be one of `bcm_switch_module_e`:

Valid Modules	Description
`bcmModuleL3EgressFec`	Modify performance of FEC APIs
`bcmModuleL3EgressEcmp`	Modify performance of ECMP APIs
`bcmModulePortMatch`	Modify performance of Port Match APIs

22.20.2 Application Reference

Example of Performance Improvement

- **Type**: CTEST reference
- **Path**: `$SDK/appl/ctest/dnx/l3/ctest_dnx_l3_performance.c`

22.21 API Descriptions

22.21.1 My-MAC Enhancements

API Name	Highlights
`bcm_switch_l3_protocol_group_set()`	Set a protocol group with a set of L3 protocols
`bcm_switch_l3_protocol_group_get()`	Get a protocol group with a set of L3 protocols
`bcm_l3_vrrp_config_add()`	Add a VRRP my MAC entry.
`bcm_l3_vrrp_config_get()`	Get a VRRP my MAC entry.
`bcm_l3_vrrp_config_delete()`	Delete a VRRP my MAC entry.
`bcm_l3_vrrp_config_delete_all()`	Delete all VRRP my MAC entries.
`bcm_l2_station_add()`	Add a multiple my MAC entry.
`bcm_l2_station_get()`	Get a multiple my MAC entry.
`bcm_l2_station_delete()`	Delete a multiple my MAC entry.

22.21.2 ETH-RIF (L3 VSI Interface) and VRF Definition

API Name	Highlights
`bcm_l3_intf_create()`	Create ETH-RIF and set some of its properties.
`bcm_l3_intf_get()`	Retrieve ETH-RIF properties from a given ETH-RIF ID.
`bcm_l3_intf_delete()`	Destroy ETH-RIF object.
`bcm_l3_intf_delete_all()`	Destroy All ETH-RIFs.
`bcm_l3_intf_find()`	Given ETH-RIF and MAC provide the ETH-RIF information.
`bcm_l3_intf_find_vlan()`	Search for L3 interface by VSI (not much different than `bcm_l3_intf_find`. Used for backward compatible purposes).
`bcm_l3_ingress_create()`	Set some of the Ingress ETH-RIF properties for a given ETH-RIF ID.
`bcm_l3_ingress_get()`	Retrieve some of the Ingress ETH-RIF properties for a given ETH-RIF ID.
`cm_multicast_l3_encap_get()`	Get the encap information from RIF type L3 interface.

22.21.3 L3 Egress Object

API Name	Highlights
`bcm_l3_egress_create()`	Create FEC/ARP-ID and set its properties.
`bcm_l3_egress_destroy()`	Destroy FEC/ARP-ID.
`bcm_l3_egress_get()`	Retrieve FEC/ARP properties given ARP-ID.
`bcm_l3_egress_arp_traverse`	Traverse all created ARP-IDs
`bcm_l3_egress_traverse()`	Traverse all created FEC-IDs.
`bcm_l3_egress_allocation_get()`	Return a range a group of consecutive unallocated FECs (used for finding FEC members for an ECMP group).
`bcm_multicast_egress_object_encap_get()`	Get the encapsulation information from egress object (FEC/ARP).

22.21.4 L3 Egress: ECMP Object

API Name	Highlights
`bcm_l3_egress_ecmp_create()`	Create an ECMP object and set its properties.
`bcm_l3_egress_ecmp_destroy()`	Destroy ECMP object.
`bcm_l3_egress_ecmp_get()`	Retrieve ECMP object with its properties given an ECMP interface ID.
`bcm_l3_egress_ecmp_add()`	Add a member to the existing ECMP group.
`bcm_l3_egress_ecmp_delete()`	Remove a member from an existing ECMP group.
`bcm_l3_egress_ecmp_traverse()`	Traverse over all created ECMP objects.
`bcm_l3_egress_allocation_get()`	Return a range a group of consecutive unallocated FECs (used for finding FEC members for an ECMP group).
`bcm_l3_egress_ecmp_tunnel_priority_set()`	Set a single ECMP tunnel priority table group members.
`bcm_l3_ecmp_tunnel_priority_map_create()`	Create a map profile between the packet tunnel priority value to the ECMP tunnel priority table.
`bcm_l3_ecmp_tunnel_priority_map_destroy()`	Destroy an existing tunnel priority mapping profile.
`bcm_l3_ecmp_tunnel_priority_map_get()`	Gets a single mapping entry between a tunnel priority value to an ECMP tunnel priority table.
`bcm_l3_ecmp_tunnel_priority_map_set()`	Sets a single mapping entry between a tunnel priority value to an ECMP tunnel priority table.

22.21.5 Route

API Name	Highlights
`bcm_l3_route_add()`	Add a route entry.
`bcm_l3_route_get()`	finds a route entry destination.
`bcm_l3_route_delete()`	Delete a route entry.
`bcm_l3_route_delete_all()`	Delete all the route entries.
`bcm_l3_route_traverse()`	Traverse over all the route entries.

22.21.6 IPMC

API Name	Highlights
`bcm_ipmc_add()`	Add an IPMC entry.
`bcm_ipmc_find()`	Finds an IPMC entry destination.
`bcm_ipmc_remove()`	Delete an IPMC entry.
`bcm_ipmc_remove_all()`	Delete all the IPMC entries.
`bcm_ipmc_traverse()`	Traverse over all the IPMC entries.
`bcm_ipmc_config_add()`	Creates a compressed_SIP or a compressed_SI entry for IPv6.
`bcm_ipmc_config_find()`	Finds a compressed entry for IPv6.
`bcm_ipmc_config_remove()`	Removes a compressed entry for IPv6.
`bcm_ipmc_config_traverse()`	Traverses all compressed entries for IPv6. Two flags are available: ■ `BCM_IPMC_CONFIG_TRAVERSE_DELETE_ALL` (all entries will be removed). ■ `BCM_IPMC_L2` (Indicates that all compressed_SIP entries will be traversed. If flag is omitted, compressed_SI entries will be traversed).

Chapter 23: MPLS Label Switch Router

23.1 Introduction

The MPLS Label Switch Router (LSR) application defines how packets are switched within an MPLS domain. This chapter elaborates on the SDK configuration for the possible actions taken following MPLS forwarding (processing one or two MPLS labels in the forwarding stage):

1. Swapping the processed label/s with a new label.
2. Popping the outermost label (PHP).
3. Swapping a label and pushing one or more labels into the MPLS stack (entering new MPLS domains).

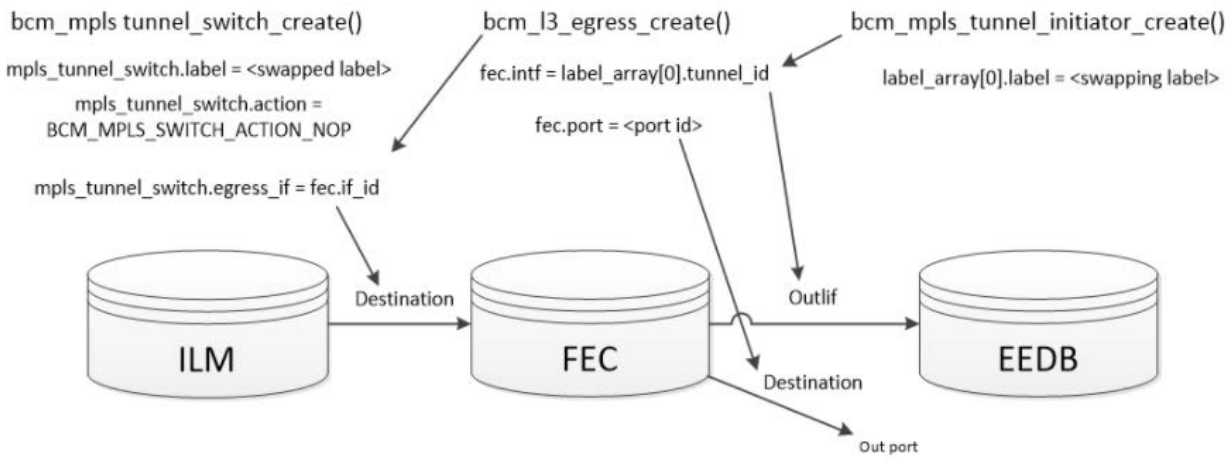
Matching an MPLS forwarding entry in the ILM is done according to one of the following two models:

- **Platform**: The matching key is a single label.
- **Per interface**: The matching key is a combination of the incoming L3 interface (for example, L3 Tunnel InLIF/ETH-RIF) with a label.

The following figures summarize the configuration of MPLS ILM application:

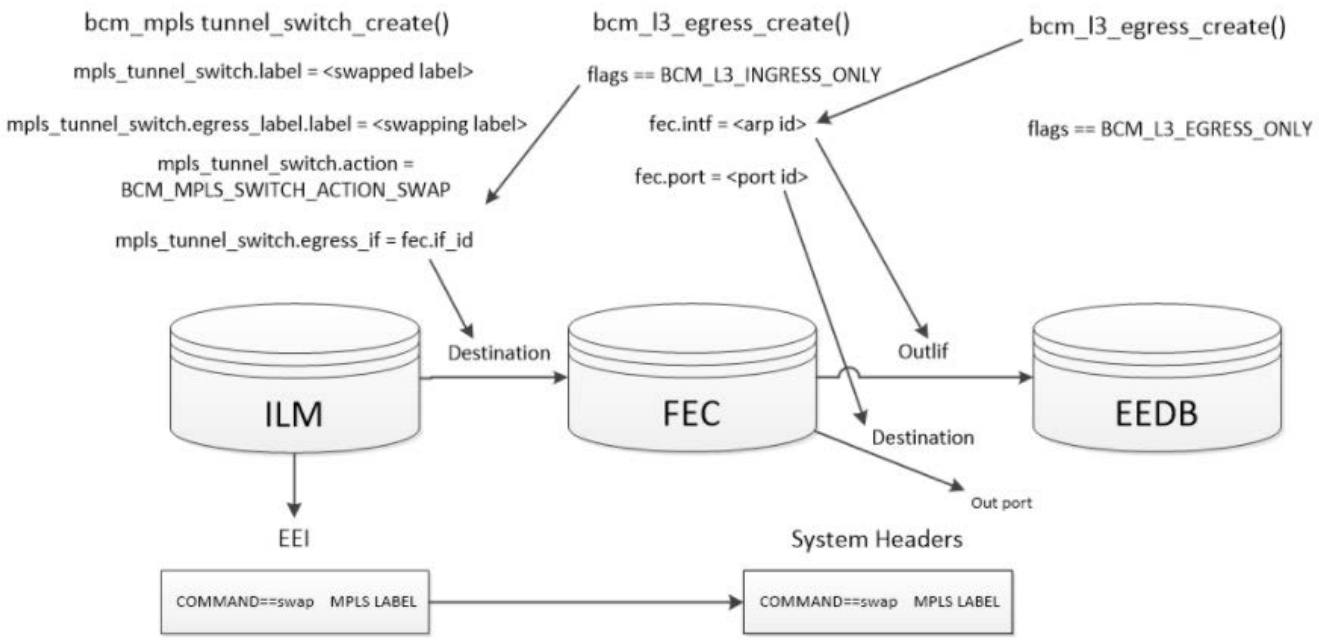
- Figure 12 shows the API configuration for an MPLS forwarding scenario with matching platform model and SWAP as the label action. The information for the swap action (i.e. the swapping label) is taken from the EEDB.
  - `bcm_mpls_tunnel_initiator_create()`: Create an entry in the EEDB with the swapping label.
  - `bcm_l3_egress_create()`: Create a FEC entry pointing to the EEDB entry.
  - `bcm_mpls_tunnel_switch_create()`: Create an ILM entry with the swapped label, pointing to the FEC entry.

Figure 12: Platform Model: Label SWAP Action from EEDB



- Figure 13 summarizes the API configuration for an MPLS forwarding scenario with matching platform model and SWAP as label action. The information for the swap action (that is, the swapping label) is taken from the ILM.
  - `bcm_l3_egress_create()` Egress-Only: Create an ARP entry in the EEDB.
  - `bcm_l3_egress_create()` Ingress-Only: Create a FEC entry pointing to the ARP entry.
  - `bcm_mpls_tunnel_switch_create()`: Create an ILM entry with the swapped label pointing to the FEC entry.

Figure 13: Platform Model: Label SWAP Action from ILM



It is recommended to read this section after reading the following sections:

- Chapter 21, Ethernet Bridge and Chapter 22, IP Router to get an overview of the basic L2 and L3 objects.
- Section 22.6, ETH-RIF (L3 VSI Interface) and VRF Definition
- Chapter 22, IP Router. This section provides information about configurations that are related to next hop information, namely FEC and Egress-ARP. Also, it provides information relevant for the My-MAC operation which is a prerequisite for MPLS tunnel forwarding and termination.
- Chapter 25, MPLS LER Encapsulation. This section is relevant for MPLS forwarding that utilizes egress resources (namely the EEDB).

23.2 Application Configuration Checklist

- Set L3 port properties (see Section 22.3, Port Routing Properties)
- Set ETH-RIF properties and create MyMac per ETH-RIF (see Chapter 22.6, ETH-RIF (L3 VSI Interface) and VRF Definition)
- Ingress MPLS termination (optional)
- Egress object: Egress-ARP (see Section 22.7, L3 Egress Object: Egress-ARP (Next-Hop))
- Egress MPLS tunnel (optional)
- Ingress object: Ingress-FEC
- MPLS forwarding: ILM
- Set fixed Remark profile for SWAP actions coming out of ILM

23.3 Ingress MPLS Termination Object

For more information regarding the configuration of MPLS entries in the Termination table, see Chapter 24, MPLS LER Termination.

23.4 Egress MPLS Tunnel

The decision of the action depends on the field `label_array[0].action` in `bcm_mpls_tunnel_initiator_create`. If `label_array[0].action == BCM_MPLS_EGRESS_ACTION_PHP` the action is PHP; otherwise it is *swap or push*. For more information, see Chapter 25, MPLS LER Encapsulation.

- Action is PHP:
  - Call `bcm_mpls_tunnel_initiator_create()`.
  - `label_array[0].flags |= BCM_MPLS_EGRESS_LABEL_ACTION_VALID | BCM_MPLS_EGRESS_LABEL_TTL_DECREMENT`.
  - `label_array[0].action = BCM_MPLS_EGRESS_ACTION_PHP`.
  - `label_array[0].l3_intf_id = <arp id>`.
- Action is SWAP:
  - The entry is configured in the same way as described in Section 25.4, Egress MPLS Tunnel.

23.5 Ingress Object: Ingress-FEC

For information about FEC creation and configuration, see Section 22.8, L3 Egress Object: Ingress-FEC.

23.6 MPLS Forwarding: ILM

ILM is the database that handles matching and forwarding resolution for MPLS LSR.

Thus, all relevant parameters are used with respect to:

- **Matching** a key, resulting with a label action. The following two matching modes are supported:
  - Platform model
  - Per-interface model.
- **Label action**: forwarding resolution per the matched label/s.

It is convenient to divide the configuration flow along these lines. Each division is further divided according to the action taken and the mode of matching (platform model or per-interface model).

All configurations are done by using the `bcm_mpls_tunnel_switch_create()` API unless indicated otherwise.

NOTE: This API is used also for configuring an MPLS LER Termination entry (Termination). For more information, see Chapter 24, MPLS LER Termination.

23.6.1 SOC Properties

None

23.6.2 Configuration Flow

To add a new ILM entry, call `bcm_mpls_tunnel_switch_create(unit, info)`:

Optional:

- `info.flags` – `BCM_MPLS_SWITCH_TTL_DECREMENT`.
  The TTL will be decremented for all MPLS forwarding related actions, even if it is inherited or copied.

NOTE: `BCM_MPLS_SWITCH_TTL_DECREMENT` is always returned in `bcm_mpls_tunnel_switch_get`.

Matching:

- Platform:
  - `info.label` – MPLS Label
- Per-Interface:
  - `info.flags` – `BCM_MPLS_SWITCH_LOOKUP_L3_INGRESS_INTF`
  - `info.label` – MPLS label
  - `info.ingress_if` – RIF/LIF ID as bcm intf type.
    - ETH-RIF created from `bcm_l3_intf_create()`.
    - LIF-ID – Last terminated MPLS-tunnel, see Section 24.3, Ingress MPLS Termination Object

Label action

Action originating from ILM:

- `info.egress_if`
  - FEC-ID encoded as `BCM_L3_ITF_TYPE_FEC`.
  - ARP encoded as `BCM_L3_ITF_TYPE_LIF`.

  Action is PHP:

  - `info.action` – `BCM_MPLS_SWITCH_ACTION_PHP or BCM_MPLS_SWITCH_ACTION_POP_DIRECT` (these flags are synonymous).
  - `info.qos_map_id` – Egress MPLS PHP QoS ID. For more information, see Section 10.11, Egress MPLS PHP QoS.
  - `info.egress_label.egress_qos_model` – PHP TTL and QoS model

    TTL only supports `bcmQosEgressModelUniform` and `bcmQosEgressModelPipeMyNameSpace`

    QOS only supports `bcmQosEgressModelUniform` and `bcmQosEgressModelPipeNextNameSpace`

  Action is SWAP

  - `info.action` – `BCM_MPLS_SWITCH_ACTION_SWAP`.
  - `info.qos_map_id` – Egress MPLS Remark QOS ID. For more information, see Section 23.7, Remark Profile for SWAP Actions Coming Out of Ingress (ILM, FEC).
  - `info.egress_label.label` – Swapping label.

Action originating not from ILM

- `info.action` – `BCM_MPLS_SWITCH_ACTION_NOP` (none).

Action originating from EEDB: See Section 25.4, Egress MPLS Tunnel. Point to the EEDB entry:

From ILM:

- `info.egress_if` – Points to Egress MPLS tunnel OutLIF encoded as `BCM_L3_ITF_TYPE_LIF`.
  - ARP
- `info.port` – Point to FEC for destination

From FEC:

- `info.egress_if` – FEC-ID encoded as BCM_L3_ITF_TYPE_FEC.Point to FEC as the destination. FEC will point to the egress MPLS tunnel.

Action originating from FEC:

- `info.egress_if` – FEC-ID encoded as BCM_L3_ITF_TYPE_FEC.
- For information regarding the definition of label action origination (info.egress_label) from FEC, see Section 22.8.2, Configuration Flow.

Action is a multicast destination.

- `info.mc_group` – Destination is MC group ID.
- `info.flags` – `BCM_MPLS_SWITCH_P2MP`.
- In this case, each replication in the multicast group provides the destination and encapsulation information (as Egress MPLS Tunnel).

Replace:

- To replace an entry in the ILM, set the same matching information (according to one of the three models) with different label action information.

Statistics:

- Enable statistics counting on the statistics interface by adding the following parameters:
  - `info.stat_id` – Statistics object ID.
  - `info.stat_pp_profile` – PP statistics ID and profile.

  For more information regarding the creation of statistics interfaces and counting, see Chapter 15, PP Statistics Generation.

23.6.3 Shell Commands

None

23.6.4 Application Reference

Per Platform configuration.

- **Type:** CINT reference.
- **Path:**
  - `src/examples/sand/cint_mpls_encapsulation_basic.c`
  - `src/examples/dnx/mpls/cint_mpls_operations_without_eedb.c`
- Action originating from EEDB:

  Main function: `mpls_encapsulation_basic_main()`.
  - SWAP: `mpls_encapsulation_basic_create_push_or_swap_tunnel()`.
  - PHP: `mpls_encapsulation_basic_create_php_tunnel()`.
- Action originating from ILM:

  Main function:

  Cases:
  - `MPLS_NO_EEDB_ILM_MODE_PHP`
  - `MPLS_NO_EEDB_ILM_MODE_SWAP`
- Action originating from FEC:

  Main function: `mpls_operations_without_eedb_main()`

  Cases:
  - `MPLS_NO_EEDB_FEC_MODE_PHP`
  - `MPLS_NO_EEDB_FEC_MODE_SWAP`
  - `MPLS_NO_EEDB_FEC_MODE_PUSH`

Per Interface configuration.

- **Type:** CINT reference.
- **Path:** `src/examples/dnx/cint_mpls_per_if.c`
- **Main function**: `mpls_per_if_main()`.

23.7 Remark Profile for SWAP Actions Coming Out of Ingress (ILM, FEC)

For more information, see Section 10.15, MPLS Ingress Swap QoS.

23.7.1 SOC Properties

None

23.7.2 Configuration Flow

- For the creation of egress remark profiles, see Chapter 10, QoS.
- For a single-swap command device, to set the global remark profile used for SWAP actions call `bcm_qos_control_set(unit, bcmQosControlMplsIngressSwapRemarkProfile, arg)`
- For multi-swap command devices, to set a remark-profile per SWAP action, add the remark-profile (qos_map_id) to the MPLS swap entry or FEC entry.

23.7.3 Application Reference

An example of MPLS Swap Remark

- **Type:** CINT reference.
- **Path:** `src/examples/sand/cint_sand_mpls_qos.c`
- **Main function:** `mpls_ingress_swap_remark_profile_set(...)`

23.8 MPLS Forwarding EXP Preserve

MPLS EXP preservation can be globally configured if egress forwarding is MPLS. If the EXP is preserved, EXP remains unchanged whenever an MPLS swap occurs or MPLS is forwarded after PHP. Otherwise, the EXP is remarked.

23.8.1 SOC Properties

None

23.8.2 Configuration Flow

To globally set egress MPLS forward EXP preserve, call the following API:

`bcm_qos_control_set( unit, 0, bcmQosControlMplsEgressForwardQoSPreserve, arg),`

– `arg` – True or False
  - True – Preserve
  - False – Remark (default)

23.9 API Descriptions

NOTE: The following APIs are also used for MPLS LER termination configurations. This description highlights the usage described in this section.

23.9.1 MPLS Forwarding: ILM

API Name	Highlights
`bcm_mpls_tunnel_switch_create()`	Create a MPLS forwarding entry in the ILM.
`bcm_mpls_tunnel_switch_get()`	Get info regarding created MPLS forwarding entry in the ILM.
`bcm_mpls_tunnel_switch_delete()`	Delete created MPLS forwarding entry in the ILM.
`bcm_mpls_tunnel_switch_delete_all()`	Delete all created MPLS forwarding entries in the ILM.
`bcm_mpls_tunnel_switch_traverse()`	Traverse all MPLS tunnels in the ingress and run a callback function provided by the user.

Chapter 24: MPLS LER Termination

24.1 Introduction

The egress MPLS label edge routing (LER) application defines how MPLS headers are terminated (pop).

Essentially, the termination process is based on two steps:

1. **Matching**: An entry is accessed according to one of the following three models:
   – **Platform**: Matching key is a single label.
   – **Per interface**: Matching key is a combination of incoming interface LIF/RIF with a label.

NOTE: The decision of match-type model is according to the following logic:
- If the incoming-interface LIF/RIF supports per-interface namespace → Per interface
- Otherwise, the platform model is decided.

2. **LIF Information retrieval**: Following an exact-match lookup, a LIF entry is accessed, resulting with information relevant for the terminated MPLS headers.

It is recommended to read this section after reading the following sections:
- Chapter 21, Ethernet Bridge and Chapter 22, IP Router to get an overview of the basic L2 and L3 objects.
- Section 22.6, ETH-RIF (L3 VSI Interface) and VRF Definition. This section provides information about configurations that are related to IP routing and next hop information, namely FEC and Egress-ARP. Also, it provides information relevant for the MyMac operation which is a prerequisite for MPLS tunnel forwarding and termination.

24.2 Application Configuration Checklist

- Set L3 port properties (see Section 22.3, Port Routing Properties)
- Set ETH-RIF properties and create MyMac per ETH-RIF (see Section 22.6, ETH-RIF (L3 VSI Interface) and VRF Definition.)
- Optional: Ingress IP Tunnel termination object (see Section 23.3, Ingress MPLS Termination Object)
- Section 24.3, Ingress MPLS Termination Object
- Special MPLS labels configuration and considerations
- MPLS FRR (Fast Reroute)

24.3 Ingress MPLS Termination Object

Ingress MPLS termination object is based on two steps, Matching and LIF Information retrieval as described in Section 24.1, Introduction.

Object creation and its main configuration is done by the `bcm_mpls_tunnel_switch_create()` API.

NOTE: This API is used also for configuring a MPLS forwarding entry (LSR). For more information, see Chapter 23, MPLS Label Switch Router.

24.3.1 SOC Properties

None

24.3.2 Configuration Flow

1. Create MPLS termination object including both Matching and Action information by calling API `bcm_mpls_tunnel_switch_create(unit, info)`:

- Mandatory:
  - `info.action` – Must be `BCM_MPLS_SWITCH_ACTION_POP` to indication termination.
- Matching:
  - Platform:
    - `info.label` – MPLS Label
  - Per-Interface:
    - `info.flags` – `BCM_MPLS_SWITCH_LOOKUP_L3_INGRESS_INTF`
    - `info.label` – MPLS label
    - `info.ingress_if` – RIF/LIF ID as bcm intf type.
      - ETH-RIF created from `bcm_l3_intf_create()`.
      - LIF-ID – Last terminated IP-Tunnel/MPLS-tunnel.
      It must be RIF/LIF. Call `bcm_l3_ingress_create()` with `ingr_itf.intf_id` == <RIF/ LIF id>, `ingr_itf.flags` == `BCM_L3_INGRESS_MPLS_INTF_NAMESPACE`. For more details, see Section 22.6.2, Configuration Flow.
  - Bud node:
    - `info.flags` – `BCM_MPLS_SWITCH2_TERM_BUD`
    - `info.label` – MPLS label for second pass termination.
      For more information, see Chapter 18, Drop-and-Continue
- LIF information
  - With ID LIF creation – If the WITH-ID option is required, fill:
    - `info.tunnel_id` – Must be of format of `BCM_GPORT_TUNNEL_ID`
      Encode the user given global LIF as a gport
    - `info.flags` – Add flag `BCM_MPLS_SWITCH_WITH_ID`.
  - Replace – If a replace is required, add:
    - `info.flags` – `BCM_MPLS_SWITCH_REPLACE`

NOTE: If the replace flag is used, the matching information of the existing object-ID cannot be changed.

  - TTL attributes – The following options exist:
    - `info.flags`:
      - `BCM_MPLS_SWITCH_TRAP_TTL_0` – Enable trap for a packet with TTL = 0.
      - `BCM_MPLS_SWITCH_TRAP_TTL_1` – Enable trap for a packet with TTL = 1.
      For more information regarding trap configuration, Chapter 12, Traps.
    - `ingress_qos_model.ingress_ttl` – Set TTL-Model (Pipe, Short-Pipe, Uniform).
      For more information regarding TTL, see Chapter 10, QoS.
  - EXP attributes – The following options exist:
    - `info.ingress_qos_model.ingress_phb`, `info.ingress_qos_model.ingress_remark` – QoS-model for PHB and Remark.
    - `info.qos_map_id` – Provide QoS-profile (PHB and Remark).

    For more information regarding EXP, see Chapter 10, QoS.

- BOS – If BOS indication required, set:
  - `info.flags`:
    - `BCM_MPLS_SWITCH_EXPECT_BOS` – This raises a trap if the received label is not BOS. For more information regarding trap configuration, see Chapter 12, Traps.
    - `BCM_MPLS_SWITCH2_EXPECT_NON_BOS` – This raises a trap if the received label is BOS.
- Protection – If protection 1+1 is required fill:
  - `info.failover_id` – Protection pointer. Created by Failover APIs.
  - `info.failover_tunnel_id` – Set to 1 if the tunnel is the primary; otherwise, it is the secondary. Note that when retrieving tunnel information, this value indicates the tunnel protection status: if 1, the tunnel is working, if 0, the tunnel is backup.

  For more information regarding the creation of a protection scheme (which produces the protection pointer), see Section 14.8, MPLS Tunnel 1+1 Protection.
- VRF:
  - `info.vpn` – Sets a VRF for this MPLS tunnel.
- Statistics:
  - `info.flags2`: `BCM_MPLS_SWITCH2_STAT_ENABLE` – Indicates that the statistics are enabled.

  Additional parameters are required for statistics. See Step 3. For more information regarding the creation of statistics interfaces and counting, Section 15, PP Statistics Generation.
- Extended-Termination:
  - `info.flags2`: `BCM_MPLS_SWITCH2_EXTENDED_TERMINATION` – Used for extended-termination flow. The tunnel must be reclassified in the second pass.
- InLIF properties:
  - `info.tunnel_if` – Configure if the object is part of the InLIF stack.

2. Set additional properties per this MPLS tunnel (Tunnel-RIF properties). Call `bcm_l3_ingress_create(unit, ing_intf, intf)`:
   - `intf` – The Tunnel ID object in Intf type encoding. The encoding can be done by calling macro `BCM_GPORT_TUNNEL_TO_L3_ITF_LIF(intf_id, mpls_term.tunnel_id)`. `mpls_term.tunnel_id` is the created object from `bcm_mpls_tunnel_switch_create` API.
   - `ing_itf.vrf` – Assigns vrf to this tunnel (same as mpls_tunnel API creation VPN field)
   - For reviewing all properties of the API, see the sub-section that discusses `bcm_l3_ingress_create` in Chapter 22, IP Router.
3. If statistics are enabled, configure their settings by calling `API bcm_gport_stat_set(unit, gport, core_id, engine_source, stat_info)` where `gport` is the MPLS-Tunnel termination object ID. For more information about the API see Chapter 15, PP Statistics Generation.
4. Configure an action-profile for the object by calling `bcm_rx_trap_lif_set`. See Section 12.5.6, LIF Traps.

24.3.3 Shell Commands

None

24.3.4 Application Reference

Per Platform configuration.

- **Type:** CINT reference.
- **Path:** `src/examples/sand/cint_mpls_termination_basic.c`
- **Main function:** `mpls_termination_basic_main()`.

Per Interface configuration.

- **Type:** CINT reference.
- **Path:** `src/examples/dnx/cint_mpls_per_if.c`
- **Main function:** `mpls_per_if_main()`

24.4 Special MPLS Labels

The termination sequence of Ingress MPLS labels described in Section 23.3, Ingress MPLS Termination Object is used for tunnel labels with a label ID above 16. Labels with a value less than 16 are considered special labels, and are described in this section.

The supported special labels are given below according to a before/after tunnel division:

- Before tunnel:
  - RA (1) – Router Alert. Used for signaling OAM is above. Used to forward the packet to the router. Not expected with BOS.
  - IPv4 Explicit (0) – Signals an IPv4 packet over LSP. Stores the relevant exp, saving the label lookup and indicating that the next header is IPv4.
  - IPv6 Explicit (2) – Signals an IPv6 packet over LSP. Stores the relevant exp, saving the label lookup and indicating that the next header is IPv6.
- After tunnel:
  - ELI (7) – Entropy Label indication, used to signal entropy over MPLS. EL label will always follow. Packets can come in format ELoELIoMPLS or ELoELIoPWE. EL is used for load balancing where the label itself is used to indicate the flow to be used, thus allowing better randomization.
  - GAL (13) – Generic Association Channel, indicates what is the next label. It is used to signal OAM type over MPLS. Packets can come in format GALoLSP or GALoPWE. Under GAL will come G-ACH the label signaling the associated channel.
  - OAM Alert Label (14) – Used to indicate OAM is above.

NOTE: No configuration is required. The above configuration takes place during the initialization sequence of the SDK.

24.5 MPLS Fast Reroute (FRR)

When a router along a route adds an extra MPLS label due to broken link, this feature enables termination of that label when it is located as the outermost label. Also note that for FRR, this feature enables the removal of the extra label that is added when packets are rerouted through a backup router.

The APIs in Section 24.6.2, FFR Termination allow for the manipulation (add/get/delete/traverse) of the list of labels on this dedicated database. These APIs relate to many kinds of *special MPLS types,* of which FRR is only one.

24.5.1 SOC Properties

None

24.5.2 Configuration Flow

1. Enable the port to allow FRR label termination by calling API `bcm_port_control_set()`:
   - Set the `port` parameter to the corresponding ingress port.
   - Set the `type` parameter to `bcmPortControlMplsFRREnable`.
   - Set the `value` parameter to 1 to enable FRR termination (or 0, to disable).

2. Add FRR label by API `bcm_mpls_special_label_identifier_add()`:
   – Set the `label_type` parameter to `bcmMplsSpecialLabelTypeFrr`.
   – Set the `label_info.label_value` parameter to the value of the MPLS label to be removed.

24.5.3 Shell Commands

None

24.5.4 Application Reference

FRR label application example:

- **Type:** CINT reference
- **Path:** `src/examples/dnx/cint_dnx_mpls_frr_basic.c`
- **Main function:** `mpls_frr_basic_main()`

24.6 API Descriptions

NOTE: The following APIs are used also for MPLS LER termination configurations. This description highlights the usage described in this section.

24.6.1 Ingress MPLS Tunnel Termination Object

API Name	Highlights
`bcm_mpls_tunnel_switch_create()`	Create a termination entry in the ingress. The indication of termination entry done by the action field (POP).
`bcm_mpls_tunnel_switch_get()`	Get info regarding created termination entry in the ingress.
`bcm_mpls_tunnel_switch_delete()`	Delete created termination entry in the ingress.
`bcm_mpls_tunnel_switch_delete_all()`	Delete all created termination entries in the ingress.
`bcm_mpls_tunnel_switch_traverse()`	Traverse all MPLS tunnels in the ingress and run a callback function provided by the user.

24.6.2 FFR Termination

API Name	Highlights
`bcm_mpls_special_label_identifier_add()`	Create a special-label-type entry. Currently only valid for FRR. In this latter case, the entry is used for ingress termination of that label.
`bcm_mpls_special_label_identifier_get()`	Get info regarding created special-label-type entry. For FRR (ingress termination), this info is only ‘exists’ or ‘does not exist’.
`bcm_mpls_special_label_identifier_delete()`	Delete created special-label-type entry.
`bcm_mpls_special_label_identifier_traverse()`	Traverse all tables corresponding to MPLS special-label-types and invoke a callback function, provided by the user, per each entry.

Chapter 25: MPLS LER Encapsulation

25.1 Introduction

The ingress MPLS label edge routing (LER) application defines how MPLS headers are encapsulated (push) following an IP/MPLS forwarding. This chapter elaborates on the actions that facilitate MPLS encapsulation.

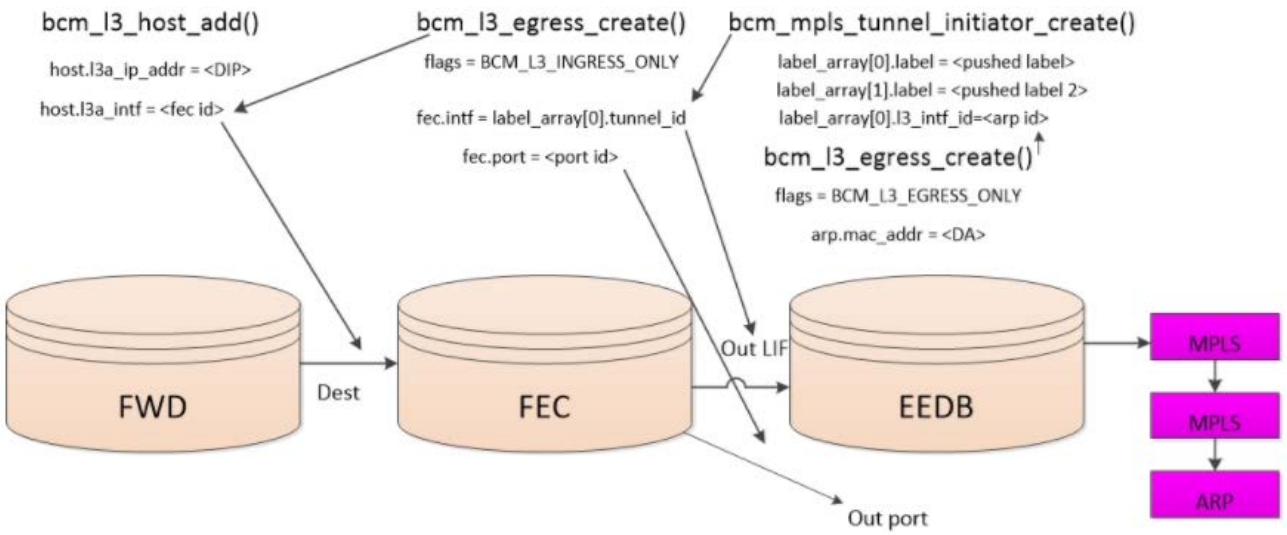
The API that configures this scheme is `bcm_mpls_tunnel_initiator_create()`. This entry contains the information relevant for the encapsulation. Each entry can provide up to two MPLS labels.

It is essential to note that this API also creates entries that are accessed following an MPLS forwarding (SWAP or PHP). The configuration of this API in the latter context is documented in Action originating not from ILM.

Figure 14 shows an example of an API configuration for an IP forwarding scenario resulting with an encapsulation of two MPLS labels:

- `bcm_l3_egress_create()`: Create an ARP entry.
- `bcm_mpls_tunnel_initiator_create()`: Create an entry in the EEDB with the MPLS labels. Point to the ARP entry.
- `bcm_l3_egress_create()`: Create a FEC entry pointing to the EEDB entry.
- `bcm_l3_host_add()`: Add an IP forwarding entry pointing to the FEC entry.

Figure 14: IP Forwarding with Two MPLS Label Encapsulations



It is recommended to read this section after reading the following sections:

- Basic bridge and IP routing sections to get an overview of the basic L2 and L3 objects.
- Section 22.6, ETH-RIF (L3 VSI Interface) and VRF Definition. This section provides information about configurations that are related to IP routing and next hop information, namely FEC and Egress-ARP. Also, it provides information relevant for the MyMac operation which is a prerequisite for IP forwarding.

25.2 Application Configuration Checklist

- Set L3 port properties (see Section 22.3, Port Routing Properties)
- Set ETH-RIF properties and create MyMac per ETH-RIF (see Section 22.6, ETH-RIF (L3 VSI Interface) and VRF Definition)
- Section 25.3, Egress Object: Egress-ARP
- Section 25.4, Egress MPLS Tunnel
- Ingress object: FEC (see Section 22.8, L3 Egress Object: Ingress-FEC)
- Forwarding configurations: IP/MPLS forwarding (see IP-Router and MPLS LSR sections)

25.3 Egress Object: Egress-ARP

An Egress-ARP object is required to hold next hop information (e.g, DA). In this scheme, this entry will be pointed from the EEDB entry (in this respect, it would be a MPLS entry) that is positioned last in the encapsulation linked list that is created in the ENCAP stage. For optimization, it is possible for ARP to directly provide the VLAN-translation information instead of having it coming from the Out-Port x RIF lookup. When this occurs, it saves one ESEM lookup and an additional entry in the egress encapsulation stack.

For information regarding the creation of this object, see Section 22.7, L3 Egress Object: Egress-ARP (Next-Hop).

25.4 Egress MPLS Tunnel

All configurations are done with the `label_array` variable passed to `bcm_mpls_tunnel_initiator_create()`. Each entry in this array reflects information in the EEDB that is relevant to one label. In order to configure an entry with two MPLS headers, set `num_labels` == 2. Else, set `num_labels` == 1. Unless indicated otherwise, each field in label_array[0] should be identical to the same field in `label_array[1]` (if user sets `num_labels` == 2). Thus, the following configuration description will be for `label_array[0]`, implying the same configuration for `label_array[1]`.

MPLS entries can be chained with consecutive calls to `bcm_mpls_tunnel_initiator_create()`. To learn how to do this, see Next EEDB entry.

NOTE: This entry may be used for either a PUSH or a SWAP command. The usage depends on the type of forwarding. Following an IP forwarding, this entry will be used for a PUSH command, meaning the label will be prepended under the IP header. Following an MPLS forwarding (after matching a label), this entry's label will swap the label that was matched in the forwarding stage.

25.4.1 SOC Properties

None

25.4.2 Configuration Flow

1. Create an egress MPLS tunnel object by calling API `bcm_mpls_tunnel_initiator_create(unit, intf, num_labels, label_array)`:

- Label assignment:
  - `num_labels` – Set the number of labels to two if two labels are required in one EEDB entry, otherwise set it to one. For two labels, `label_array[0]` is the first label and `label_array[1]` is the second label.
- With ID LIF creation:
  - `BCM_L3_ITF_SET(label_array[0]->tunnel_id)`
    - Encode the user given global LIF
  - `label_array[0].flags |= BCM_MPLS_EGRESS_LABEL_WITH_ID.`
  - Replace:
    - `info.flags |= BCM_MPLS_EGRESS_LABEL_REPLACE.`
    - `label_array[0].tunnel_id` – Represents an existing LIF.
- Without global LIF creation (can be used when the object is only pointed by other EEDB object link-list and not from ingress/ACL/MC tables):
  - `label_array[0].flags |= BCM_MPLS_EGRESS_LABEL_VIRTUAL_EGRESS_POINTED`
  - `BCM_GPORT_SUB_TYPE_VIRTUAL_EGRESS_POINTED_SET(label_array[0]` → `tunnel_id).`
- Next EEDB entry
  - `label_array[0].l3_intf_id` – Connect this entry to one of the following EEDB entries:
    - ARP-ID (from `bcm_l3_egress_create` `EGRESS_ONLY` creation), encoded as `BCM_L3_ITF_TYPE_LIF`.
    - Next IP/MPLS Tunnel-ID encoded as `BCM_L3_ITF_TYPE_LIF`.
    - No connection to another EEDB entry: Set to `BCM_IF_INVALID`.

    By default, the value is set to 0, which means reserve the next EEDB pointer in the entry for future use (the value can be changed upon replace).
- TTL attributes:
  - `label_array[0].flags`:
    - `BCM_MPLS_EGRESS_LABEL_TTL_DECREMENT` – Indicate TTL decrement is done, must be set.
  - `label_array[0].ttl` – The TTL value used if Pipe TTL model is set.
  - `label_array[0].egress_qos_model.egress_ttl` – TTL model.

  For more information about QoS Egress TTL, see Section 10.18, API Descriptions.
- EXP attributes:
  - `label_array[0].flags`:
    - `label_array[0].exp` – The EXP value (for the case of Pipe).
    - `label_array[0].qos_map_id` – The egress remark QoS profile.
    - `label_array[0].egress_qos_model.egress_qos` – EXP model.

  For more information about QoS Remark mapping, see Chapter 10, QoS.
- Protection: To configure protection (1:1 multicast) for this tunnel set:
  - `label_array[0].flags` – `BCM_MPLS_EGRESS_LABEL_PROTECTION`.
  - `label_array[0].egress_failover_if_id` – Set to 1 if this tunnel is the primary one. A value of 0 indicates this is the secondary tunnel.
  - `label_array[0].egress_failover_id` – Protection pointer created by failover APIs.

  For more information regarding the creation of a protection scheme (which produces the protection pointer), see Section 14.7, MPLS-Port 1:1 Multicast Protection.

- Additional headers:
  - Entropy label addition:
    - `label_array[0].flags` – `BCM_MPLS_EGRESS_LABEL_ENTROPY_ENABLE`.
      The value of the entropy label is calculated from the LB-key and is built as {2b'2, LB-Key (18bH)}
  - Entropy label indication label addition prior to the entropy label:
    - `label_array[0].flags` – `BCM_MPLS_EGRESS_LABEL_ENTROPY_INDICATION_ENABLE`.
      The value of the ELI label is 7.

NOTE: Second flags **must** be set if the first flags are set. If `num_labels` == 2, EL + ELI are set for BOS label only.

- Tandem:
  - `label_array[0].flags` – `BCM_MPLS_EGRESS_LABEL_TANDEM`.
    - Indicates that this MPLS entry can be processed in the same Encap stage as the preceding MPLS entry (up to two entries in one stage).
- Encap Access:
  - `label_array[0].encap_access` – Assign one out of seven potential MPLS phases to this MPLS tunnel:
    - `bcmEncapAccessRif` (For deep stack).
    - `bcmEncapAccessNativeArp` (For deep stack).
    - `bcmEncapAccessTunnel1`.
    - `bcmEncapAccessTunnel2`.
    - `bcmEncapAccessTunnel3`.
    - `bcmEncapAccessTunnel4`.
    - `bcmEncapAccessArp` (For deep stack).

    For more information about encapsulation phases see Section 7.4, EEDB Management.

NOTE: If the MPLS tunnel is located in encap access of ARP, then an ARP + AC entry should be created (ARP with VLAN translation) by calling `bcm_l3_egress_create()` with `BCM_L3_FLAGS2_VLAN_TRANSLATION` flag. For more information regarding the creation of ARP, see Chapter 22, IP Router.

- Statistics:
  - `label_array[0].flags` – `BCM_MPLS_EGRESS_LABEL_STAT_ENABLE`. Indicates statistics are enabled
    For more information regarding the creation of statistics interfaces and counting, see Chapter 15, PP Statistics Generation.

2. To set a MTU profile per egress MPLS tunnel, see Section 12.5.4, ETPP MTU Traps.

25.4.3 Shell Commands

None

25.4.4 Application Reference

Simple MPLS encapsulation.

- **Type:** CINT reference.
- **Path:** `src/examples/sand/cint_mpls_encapsulation_basic.c`
- **Main function:** `mpls_encapsulation_basic_main()`

Deep MPLS stack + Tandem + Encap Access.

- **Type:** CINT reference.
- **Path:** `src/examples/dnx/cint_mpls_deep_stack.c`
- **Main function:** `mpls_deep_stack_example()`

MPLS Egress Protection

- **Type:** CINT reference.
- **Path:** `src/examples/sand/cint_mpls_3_level_protection.c`
- **Main function:** `mpls_3_level_protection_basic_main()`

25.5 API Descriptions

25.5.1 Egress MPLS Tunnel

API Name	Highlights
`bcm_mpls_tunnel_initiator_create()`	Create MPLS tunnels in the egress for with push, swap or php action.
`bcm_mpls_tunnel_initiator_get()`	Get info regarding MPLS tunnels in the egress.
`bcm_mpls_tunnel_initiator_clear()`	Delete MPLS tunnels in the egress.
`bcm_mpls_tunnel_initiator_clear_all()`	Delete all MPLS tunnels in the egress.

Chapter 26: VPLS and VPWS

26.1 Introduction

VPLS and VPWS are L2 VPN applications that are configured over an MPLS infrastructure. The underlying forwarding method is Ethernet bridging. An Ethernet packet is bridged into an encapsulation that includes a label (PWE) over an MPLS stack. The packet travels across the MPLS cloud and is bridged outwards following a termination of the PWE label and the underlying MPLS stack. The following discussion revolves around the API `bcm_mpls_port_add()` which is responsible for adding an ingress PWE object (for VPLS/VPWS termination), as well as an egress PWE object (for encapsulation).

NOTE: Ingress and Egress PWE objects are created in *separate* calls: one for the ingress object, the other for the egress object.

This section includes configurations relevant to VPLS (MP) and VPWS (P2P). It focuses on the PWE object and core-side configuration (for both termination and encapsulation sides). For more information on the access-side configuration, see Chapter 27, Q-in-Q Bridging.

Figure 15 shows an example of Multipoint Ethernet bridging with PWE Encapsulation (with learning) where forwarding is done using the MACT table:

- `bcm_l3_egress_create()`: Create an ARP entry.
- `bcm_mpls_tunnel_initiator_create()`: Create an entry in the EEDB with an MPLS label. Point to the ARP entry.
- `bcm_mpls_port_add()`: Egress-only: Create an entry in the EEDB with a PWE label.
- `bcm_l3_egress_create()`: Create a FEC entry pointing to the MPLS EEDB entry.
- `bcm_mpls_port_add()`: Ingress-only: Create a termination entry for the PWE label with learning information: the global lif of the above created PWE entry and the FEC pointing to the above created MPLS entry.

Figure 15: Multipoint Ethernet Bridging with PWE Encapsulation (with Learning)

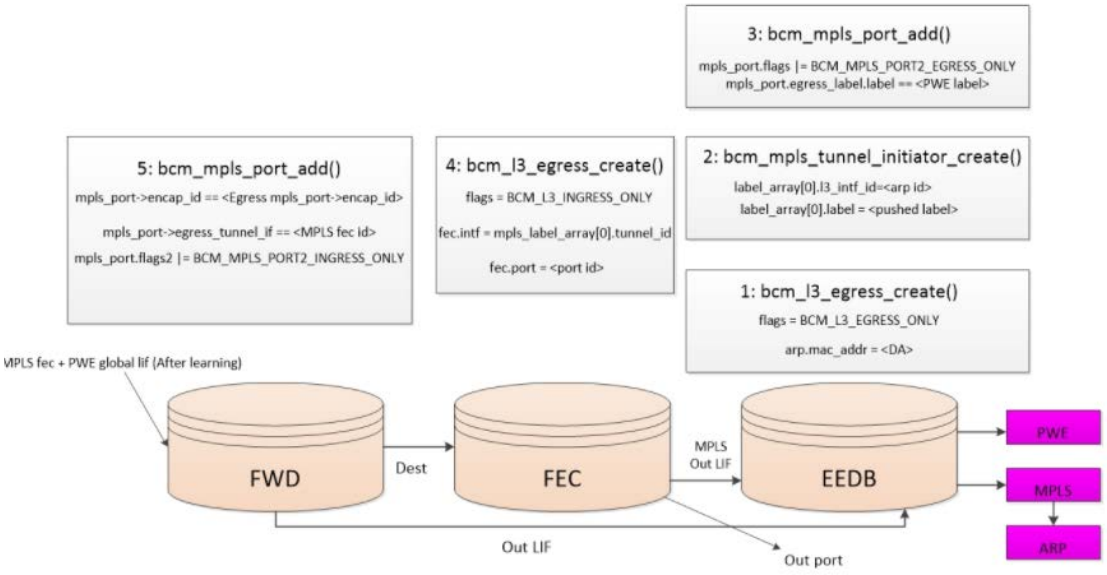
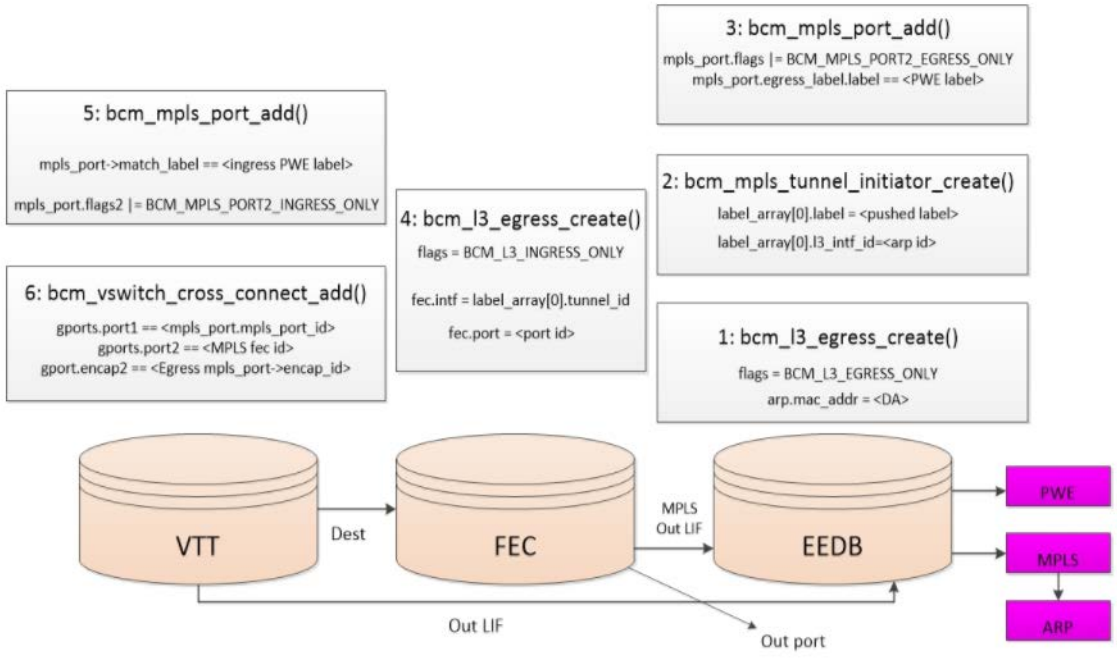


Figure 16 shows an example of P2P (Point to Point) connection of two PWEs:

- `bcm_l3_egress_create()`: Create an ARP entry.
- `bcm_mpls_tunnel_initiator_create()`: Create an entry in the EEDB with an MPLS label. Point to the ARP entry.
- `bcm_mpls_port_add()`: Create an entry in the EEDB with a PWE label.
- `bcm_l3_egress_create()`: Create a FEC entry pointing to the MPLS EEDB entry.
- `bcm_mpls_port_add()`: Create a (P2P) termination entry for the PWE label.
- `bcm_vswitch_cross_connect_add()`: Cross connect the two PWEs: Set the LIF entry of the ingress PWE with outlif == <gport of egress PWE>, destination == <FEC pointing to the MPLS entry>.

Figure 16: P2P PWE to PWE



It is recommended to read this chapter after the following chapters and sections:

- Chapter 21, Ethernet Bridge and Chapter 22, IP Router to get an overview of the basic L2 and L3 objects.
- Section 22.6, ETH-RIF (L3 VSI Interface) and VRF Definition. This section provides information about configurations that are related to IP routing and next hop information, namely FEC and Egress-ARP. Also, it provides information relevant for the MyMac operation which is a prerequisite for VPLS termination.
- Chapter 21, Ethernet Bridge. This chapter provides information about configurations that are related to Ethernet Bridging. These configurations are relevant both to bridging into and out of a L2VPN tunnel (such as VPLS).
- Section 21.12, MACT Management: L2 Learn. This section provides information regarding learning that is performed as part of a L2 service. This is relevant both to bridging into and out of a VPLS tunnel.
- Chapter 25, MPLS LER Encapsulation. This chapter provides information regarding the encapsulation of MPLS labels.
- Chapter 24, MPLS LER Termination. This chapter provides information regarding the termination of MPLS labels.

26.2 Application Configuration Checklist

- Global Configuration
- Push-Profile Allocation (when working in BCM88670 interoperability mode). For more information refer to Section 5.12, Push-Profile Allocation.
- Set L3 port properties (see Section 22.3, Port Routing Properties)
- L2 VPN (VSI) Configuration for MPLS-Port configuration
- Set ETH-RIF properties and create MyMac per ETH-RIF (see Section 22.6, ETH-RIF (L3 VSI Interface) and VRF Definition)
- Egress-Side:
  - Egress object: Egress-ARP (see Section 22.7, L3 Egress Object: Egress-ARP (Next-Hop))
  - Ingress object: FEC for MPLS tunnel and Egress MPLS tunnel (see Chapter 25, MPLS LER Encapsulation)
  - Egress object: MPLS-Port PWE
- Ingress-Side:
  - Ingress MPLS termination object (see Chapter 24, MPLS LER Termination)
  - Ingress object: MPLS-Port PWE
- Forwarding-information:
  - Add MPLS-Port PWE to flooding group
  - Forwarding configurations for VPLS: Ethernet Bridging
  - Forwarding configurations for VPWS: Cross Connect
- Protection MPLS-Port PWE

26.3 Global Configuration

The following global configuration is required for PWE object:

Set the control-word value for the egress object MPLS-Port PWE: call `bcm_switch_control_set(unit, type,arg)`

- `type` – `bcmSwitchMplsPWControlWord`.
- `arg` – Value of the control word.

26.4 L2 VPN (VSI) for MPLS-Port Configuration

Call `bcm_mpls_vpn_id_create(unit,&vpn_info)`:

- `vpn_info.flags = BCM_MPLS_VPN_VPLS | BCM_MPLS_VPN_WITH_ID.`
- `vpn_info.vpn = vpn.`
- `vpn_info.unknown_unicast_group = Multicast-ID.`
- `vpn_info.unknown_multicast_group = vpn.`
- `vpn_info.broadcast_group = vpn.`

The creation of the multicast groups is done with `bcm_multicast_create(unit, flags, mc_group_id)`:

- `flags = BCM_MULTICAST_INGRESS_GROUP | BCM_MULTICAST_WITH_ID.`
- `mc_group_id = vpn.`

26.5 Egress Object: MPLS-Port PWE

The configuration of Egress-Object MPLS-Port PWE is done at the egress device using BCM API `bcm_mpls_port_add()` with the `EGRESS_ONLY` flag. The egress object MPLS-Port PWE is responsible for the encapsulation of the PWE label as well as the next pointer information for an additional encapsulation stack, if required. The Egress-Object MPLS-Port PWE allocates an OutLIF object encoded in the MPLS-Port object (`mpls_port_id`).

NOTE: It is not possible to replace certain attributes of the egress-object PWE. For example, it is not possible to replace an unprotected PWE with a protected PWE and vice versa.

26.5.1 SOC Properties

None

26.5.2 Configuration Flow

1. Create MPLS-Port by calling `bcm_mpls_port_add(unit, vpn, mpls_port)`:

- Mandatory:
  - `mpls_port.flags2` |= `BCM_MPLS_PORT2_EGRESS_ONLY.`
  - `mpls_port.flags` |= `BCM_MPLS_PORT_EGRESS_TUNNEL.`
  - `mpls_port.egress_label.flags` |= `BCM_MPLS_EGRESS_LABEL_TTL_DECREMENT.`
  - `vpn` – Must be set to 0.
- With ID LIF creation:

  Set the user given global lif in one of two (mutually exclusive) ways:

  a. Using `mpls_port_id`:
     - `mpls_port.flags` |= `BCM_MPLS_PORT_WITH_ID.`
     - Encode the user given global LIF with:

       `BCM_GPORT_SUB_TYPE_LIF_SET(gport,BCM_GPORT_SUB_TYPE_LIF_EXC_EGRESS_ONLY, global_lif).`

       `BCM_GPORT_MPLS_PORT_ID_SET(mpls_port.mpls_port_id, gport).`

  b. Using `encap_id`:
     - `mpls_port.flags` |= `BCM_MPLS_PORT_ENCAP_WITH_ID.`
     - `mpls_port.encap_id` – User given Global-LIF.

  - Symmetric Allocation: In most cases Egress object PWE and Ingress object PWE share the same Global-LIF-ID. This is useful for Same-interface filtering. To ease on symmetric allocation using two MPLS-Port APIs (one for Egress and one for Ingress) call:
    - `mpls_port.flags2` |= `BCM_MPLS_PORT2_SYMMETRIC_ALLOC` (Using this flag verifies that the allocated global LIF is available for symmetric use i.e. available on both Egress and Ingress sides).

      Note: The next call that creates the other side of the symmetric global LIF should be allocated with BCM_MPLS_PORT_WITH_ID, and that those calls should be consecutive. This flag cannot be called with `BCM_MPLS_PORT_REPLACE`.
  - Replace:
    - `mpls_port.flags` |= `BCM_MPLS_PORT_REPLACE`.
    - Given mpls_port.mpls_port_id is for an existing LIF.
    - This works only with Item a from the previous step.
- Label assignment: Set PWE label value:
  - `mpls_port.egress_label.label` – PWE label.
- Skip TPID-class: Skip LLVP Class field assignment. When set, the default TPID-class is used.
  - `mpls_port.flags2 |= BCM_MPLS_PORT2_NO_TPID_CLASS`
- Next EEDB entry: PWE object can point to next EEDB entry by setting:
  - `mpls_port.egress_tunnel_if` – Next Global-LIF ID.
    - This entry can point to either a MPLS entry, an ARP entry or to no other EEDB entry.
- TTL attributes:
  - For TTL model set:
    - `mpls_port.egress_label.egress_qos_model.egress_ttl` – The TTL model, which is `bcmQosEgressModelNextNameSpace` for Pipe.
    - `mpls_port.egress_label.ttl` – TTL value.

    The default uniform model is selected. For more information refer to Section 10.18, API Descriptions.
- EXP attributes:
  - For EXP model set:
    - `mpls_port.egress_label.egress_qos_model.egress_qos` – The QoS model, which is `bcmQosEgressModelNextNameSpace` for Pipe.
    - `mpls_port.egress_label.exp` – EXP value.

    The default uniform model is selected. For more information see Chapter 10, QoS.
- Remark profile:
  - `mpls_port.egress_label.qos_map_id` – Remark QoS profile.

  For more information about QoS mapping, Remark profile, see Chapter 10, QoS.
- Protection 1:1 Multicast:
  - `mpls_port.flags2 |= BCM_MPLS_PORT2_EGRESS_PROTECTION` – Indicate PWE is protected.
  - `mpls_port.egress_failover_port_id` – Set to 1 if PWE object is the primary object.
  - `mpls_port.egress_failover_id` – The failover protection pointer

  For more information regarding the creation of a protection scheme (which produces the protection pointer), see Section 14.7, MPLS-Port 1:1 Multicast Protection. It is not possible to replace a protected PWE with a non-protected PWE and vice versa.

- Additional headers:
  - `mpls_port.egress_label.flags` |= `BCM_MPLS_EGRESS_LABEL_ENTROPY_ENABLE`.
    - Adds an entropy label above this PWE tunnel.
  - `mpls_port.egress_label.flags` |= `BCM_MPLS_EGRESS_LABEL_ENTROPY_INDICATION_ENABLE`.
    - Adds an entropy label indication label prior to the entropy label.
  - `mpls_port.flags` |= `BCM_MPLS_PORT_CONTROL_WORD`.
    - Adds a control word above this PWE tunnel.
    - The Control value is global in the device see Section 26.3, Global Configuration.
- Encap Access:
  - Assign one out of two potential VPLS phases to this PWE tunnel:
    `mpls_port.egress_label.encap_access` – `bcmEncapAccessTunnel`<ordered number of MPLS phase>:
    - `bcmEncapAccessTunnel1`.
    - `bcmEncapAccessTunnel2`.

    If `bcmEncapAccessInvalid` is used then it is the same as setting `bcmEncapAccessTunnel1`.

    For more information about Encap Access, see Section 7.4, EEDB Management.

NOTE: It is not possible to replace the EncapAccess field after creation.

- Split Horizon: Set an orientation for this PWE object:
  - `mpls_port.network_group_id` – Network Group ID.

  For more information regarding split horizon, see Section 20.2, Split Horizon (Network Groups).

NOTE: To correctly utilize the network group ID, follow the steps in Section 20.2.4, Configure Split Horizon for Egress L2 VPN Tunnels (PWE, VXLAN, EVPN).

- Statistics: Enable statistics on PWE object by calling:
  - `mpls_port.egress_label.flags |= BCM_MPLS_EGRESS_LABEL_STAT_ENABLE` – Indicates statistics are enabled

  For more information regarding the creation of statistics interfaces and counting, see Chapter 15, PP Statistics Generation.

2. Configure MPLS-Port statistics (if enabled) by calling `bcm_gport_stat_set`. For more information regarding the creation of statistics interfaces and counting, see Chapter 15, PP Statistics Generation.
3. Connect MPLS-Port to the Native AC entry. It is required to align the network group ID of the Native AC entry with the MPLS-Port. This is done by bcm_port_match_add API. See more information in Section 20.2.4, Configure Split Horizon for Egress L2 VPN Tunnels (PWE, VXLAN, EVPN).

26.5.3 Shell Commands

None

26.5.4 Application Reference

Simple MP scenario (no learning).

- **Type:** CINT reference.
- **Path:** `src/examples/sand/cint_vpls_mp_basic.c`
- **Main function:** `vpls_mp_basic_main()`.

Simple MP scenario (with learning).

- **Type:** CINT reference.
- **Path:** `src/examples/sand/utility/cint_sand_utils_vpls.c`
- **Main function:** `vpls_main()`.

Simple P2P scenario

- **Type:** CINT reference.
- **Path:** `src/examples/sand/cint_vswitch_cross_connect_p2p_ac_pwe.c`
- **Main function:** `vswitch_cross_connect_p2p_ac_pwe_with_ports__main_config__start_run()`.

Split Horizon

- **Type:** CINT reference.
- **Path:** `src/examples/sand/cint_vpls_split_horizon.c`
- **Main function:** `vpls_split_horizon_main()`.

26.6 Ingress Object: MPLS-Port PWE

Configuration of Ingress-Object MPLS-Port PWE is done at the ingress device using the BCM API `bcm_mpls_port_add()` with the `INGRESS_ONLY` flag. The ingress object MPLS-Port PWE is responsible for the termination of the PWE label as well as the learning information for forwarding tables. The Ingress-Object MPLS-Port PWE allocates an InLIF object encoded in MPLS-Port object (`mpls_port_id`).

The termination process is based on two steps:

1. **Matching:** An entry is accessed according to one of the following three models (this is similar to MPLS label termination. For information, see Chapter 24, MPLS LER Termination):
   - **Platform:** Matching key is a single label.
   - **Per interface:** Matching key is a combination of incoming interface LIF/RIF with a label.

NOTE: The decision of match-type model is according to the following logic:

- If the incoming-interface LIF/RIF supports per-interface namespace – Per interface
- Otherwise, Platform model is decided.

2. **LIF Information retrieval:** A LIF entry is accessed, resulting with information relevant for the terminated PWE headers.

NOTE:

- It is not possible to replace certain attributes of the Ingress-object PWE. For example, it is not possible to replace an unprotected PWE with a protected PWE, and vice versa.
- To configure the same interface filter, create the ingress PWE with a given ID (the same as the egress PWE ID).

26.6.1 SOC Properties

None

26.6.2 Configuration Flow

1. Create MPLS-Port by calling `bcm_mpls_port_add(unit, vpn, mpls_port)`:
   - Mandatory:
     - `mpls_port.flags2 |= BCM_MPLS_PORT2_INGRESS_ONLY`
     - `mpls_port.flags |= BCM_MPLS_PORT_EGRESS_TUNNEL`
     - `mpls_port.criteria =`

       `BCM_MPLS_PORT_MATCH_LABEL` – Indicates that the match criteria is label only)

       `BCM_MPLS_PORT_MATCH_LABEL_L3_INGRESS_INTF` – Indicates that the match criteria is label and preceding RIF/LIF ID
     - Indicate whether this PWE object is used Point-to-Point (P2P) or Multipoint (MP):

       `mpls_port.flags2 |= BCM_MPLS_PORT2_CROSS_CONNECT` (PWE is P2P)

NOTE: It is not possible to replace a PWE P2P with a MP PWE, and vice versa.

     - `mpls_port.qos_map_id` – Set to 0. For this application, the QoS map id should be configured using `bcm_qos_port_map_set()`. For more information about Ingress QoS mapping, see Chapter 10, QoS.
   - With ID LIF creation:
     - To provide the Global-LIF, encode the user-given global LIF with:

       `BCM_GPORT_SUB_TYPE_LIF_SET(gport,BCM_GPORT_SUB_TYPE_LIF_EXC_INGRESS_ONLY, global_lif).`

       `BCM_GPORT_MPLS_PORT_ID_SET(mpls_port.mpls_port_id, gport).`

       `mpls_port.flags |= BCM_MPLS_PORT_WITH_ID.`
     - Replace:

       `mpls_port.flags |= BCM_MPLS_PORT_REPLACE.`

       `mpls_port.mpls_port_id` – The existing LIF from previous creation.
   - VPN:
     - `vpn` – Assign VPN (VSI) to this PWE. See VPN creation in Section 26.4, L2 VPN (VSI) for MPLS-Port Configuration.
   - Label assignment:
     - `mpls_port.match_label` – This is the terminated label that is associated with this PWE object.
     - `mpls_port.context_label` – This is the MPLS label value preceding the VC label. Used in case the matching criteria is `BCM_MPLS_PORT_MATCH_LABEL_CONTEXT_LABEL`
   - Context interface:
     - `mpls_port.ingress_if` – This is the terminated LIF/RIF ID in the interface immediately preceding the VC label.
   - TTL attributes:
     - `mpls_port.ingress_qos_model.ingress_ttl` – Set TTL-Model (Pipe, Short-Pipe, Uniform). For more information regarding TTL, see Section 10.7, Ingress TTL.
     - `mpls_port.vccv_type` – Set with `bcmMplsPortControlChannelTtl` to trap a packet with TTL = 0 / 1 (similar as Reject TTL 0/1 in MPLS-Tunnel). For more information regarding trap configuration, see Chapter 12, Traps.

- EXP attributes:
  - `info.ingress_qos_model.ingress_phb`, `info.ingress_qos_model.ingress_remark` – QoS-model for PHB and Remark.
- Protection 1+1:
  - `mpls_port.ingress_failover_port_id` – Set to 1 if the PWE object is the primary object.
  - `mpls_port.ingress_failover_id` – The failover protection pointer.

  For more information regarding the creation of a protection scheme (which produces the protection pointer), see Section 14.6, MPLS-Port 1+1 Protection.

NOTE: It is not possible to replace a protected PWE with a non-protected PWE and vice versa.

- Split Horizon:
  - `mpls_port.network_group_id` – Network group ID. Orientation for this PWE object.

  For more information regarding split horizon, see Section 20.2, Split Horizon (Network Groups).
- Learning or Forwarding P2P information: Provide information to associate this PWE object with the SA of the exposed Ethernet header (for MP) or information relevant to P2P forwarding to the egress object that is associated with this PWE:
  - `mpls_port.encap_id` – The Egress PWE Global-LIF.
  - `mpls_port.egress_tunnel_if` – The MPLS FEC ID.
  - `mpls_port.port` – The physical port destination.
  - `mpls_port.failover_mc_group` – The multicast group that was configured as part of a 1+1 protection scheme. See Section 26.11, Protection PWE MPLS-Port.
  - `mpls_port.failover_port_id` – The PWE FEC-ID that was configured as part of a 1:1 protection scheme. See Section 26.11, Protection PWE MPLS-Port.
  - `mpls_port.learn_strength` – Learn strength in L2 learned entry. The field is replaceable.
- Entropy label and Entropy label indication label:
  - `mpls_port.flags |= BCM_MPLS_PORT_ENTROPY_ENABLE.`

    Assumes that an Entropy label is expected above this PWE label, with no Entropy label indication (ELI) label preceding it
- Control word:
  - `mpls_port.flags |= BCM_MPLS_PORT_CONTROL_WORD.`

    Control word is expected above this PWE label.
- Statistics: Enable statistics on PWE object by calling:
  - `mpls_port.flags2 |= BCM_MPLS_PORT2_STAT_ENABLE` – Indicates statistics are enabled

    For more information regarding the creation of statistics interfaces and counting, see Chapter 15, PP Statistics Generation.

2. Configure MPLS-Port Remark and PHB profile by calling `bcm_qos_map_add`.

   For more information about Ingress QoS mapping, see Chapter 10, QoS.
3. Configure MPLS-Port statistics (if enabled) by calling `bcm_gport_stat_set`. For more information regarding the creation of statistics interfaces and counting, see Chapter 15, PP Statistics Generation.
4. Configure MPLS-Port wide data information by calling `bcm_port_wide_data_set`. See Section 26.7, MPLS-Port PWE Wide Data.

26.6.3 Shell Commands

None

26.6.4 Application Reference

Simple MP scenario (no learning).

- **Type:** CINT reference.
- **Path:** `src/examples/sand/cint_vpls_mp_basic.c`
- **Main function:** `vpls_mp_basic_main()`.

Simple MP scenario (with learning).

- **Type:** CINT reference.
- **Path:** `src/examples/sand/utility/cint_sand_utils_vpls.c`
- **Main function:** `vpls_main()`.

Simple P2P scenario:

- **Type:** CINT reference.
- **Path:** `src/examples/sand/cint_vswitch_cross_connect_p2p_ac_pwe.c`
- **Main function:** `vswitch_cross_connect_p2p_ac_pwe_with_ports__main_config__start_run()`.

Split Horizon:

- **Type:** CINT reference.
- **Path:** `src/examples/dnx/cint_vpls_split_horizon.c`
- **Main function:** `vpls_split_horizon_main()`

26.7 MPLS-Port PWE Wide Data

Wide data can support MPLS-Port Ingress LIF up to 8 bits.

NOTE: The BCM88690 can support MPLS-Port (PWE) and IP-tunnels wide data by global configuration using `bcm_switch_control_set(type: bcmSwitchInLifPweAndIPTunnelWideDataAdjustEnable)`.
When enabled, the wide data overrides the EEI signal in the ingress termination stages (VTT). Therefore, any EEI cannot be used on any In-LIF.

26.7.1 SOC Properties

None

26.7.2 Configuration Flow

Set wide data for PWE by calling `bcm_port_wide_data_set(unit, gport, flag, wide_data)` with:

- `gport` = PWE gport
- `flag` = BCM_PORT_WIDE_DATA_INGRESS
- `wide_data`, up to 8 bits.

Create an ACL rule on the ingress PMF that contains PWE wide data as a qualifier:

- Use `bcmFieldQualifyPweInLifWideData` as a predefined qualifier
- A user-defined qualifier is allowed for wide data.

26.7.3 Shell Commands

None

26.7.4 Application Reference

PWE wide data example:

- **Type**: CINT reference.
- **Path**: `src/examples/dnx/cint_inlif_wide_data.c`
- **Main function**: `inlif_wide_data_general_service()`

26.8 Forwarding Configurations for VPLS: Ethernet Bridging (Multipoint)

To associate a static MAC DA with L2VPN LIF MPLS-Port PWE, see Section 21.14, L2 MACT Forwarding.

26.9 Forwarding Configurations for VPWS: Cross Connect

To connect an ingress PWE object in a P2P (cross-connect) connection with an egress PWE or AC object, call `bcm_vswitch_cross_connect_add()` with port1 as the ingress object:

- PWE Raw mode (default case): MPLS Port PWE created `mpls_port_id`

26.10 Adding PWE to a Flooding Group (Multipoint)

The steps in this section show how to add PWE to a flooding group of the created VPN multicast-ID.

1. Get Encap-ID from the egress object: MPLS-Port PWE: `bcm_multicast_vpls_encap_get(unit, multicast_group, port, mpls_port_id, encap_id)`:
   - `mpls_port_id` – The Egress MPLS-Port PWE object.
   - `encap_id` – The returned encap-ID to be used by the multicast group.
2. Add the object to multicast-group:
   - If the replication multicast entry includes physical-port and one OutLIF (only PWE), only then add PWE directly using the `bcm_multicast_add` API. The port is the PWE physical-port, and `encap_id` is the object from the first step.
   - If the replication multicast entry includes more than one OutLIF (for example, PWE and MPLS-Tunnel) then egress encapsulation extension is required. Follow the steps as in Chapter 4, MC Encapsulation Extension.

NOTE:

- If PWE is 1:1 Multicast protection, two entries for the PWE should be added: one with the primary and the other with the secondary.

26.11 Protection PWE MPLS-Port

The MPLS-Port can be protected as follows:

- For 1:1 unicast protection, it is expected to create an FEC object. FEC points to both Egress-Object PWEs (one for primary and one for secondary). Ingress-Object PWEs will learn the protected-FEC. FEC will be used as the main object for the forwarding cases (MP and P2P).
- For 1:1 multicast protection, it is expected that the Egress-Object PWEs will be protected and indicate which object is primary and which is secondary (accordingly with the unicast case).
- For 1+1 protection, it is expected to create two Ingress-Object PWEs. One will be primary and the other is secondary. The learning in this case is Multicast-group.

For full sequence of different protection cases see Chapter 14, Protection Switching.

26.12 API Descriptions

API Name	Highlights
`bcm_mpls_port_add()`	Create PWE object either in ingress or egress.
`bcm_mpls_port_get()`	Get info regarding a PWE object either in ingress or egress.
`bcm_mpls_port_delete()`	Delete PWE object either in ingress or egress.
`bcm_mpls_port_traverse()`	Traverse/delete over PWE ingress and/or egress objects. Traverse/delete over ingress of certain VPNs is supported. When the callback function is NULL and `traverse_flags` is `BCM_MPLS_TRAVERSE_DELETE`, it indicates deleting ingress and/or egress objects; otherwise, it indicates traversing PWE ingress and/or egress objects.
`bcm_multicast_vpls_encap_get()`	Get Encap-ID used for multicast group.
`bcm_mpls_vpn_id_create()`	Create an MPLS VPN with configuration.
`bcm_mpls_vpn_id_destroy()`	Destroy an MPLS VPN.
`bcm_mpls_vpn_id_get()`	Get existing MPLS VPN information.
`bcm_mpls_vpn_traverse()`	Traverse all MPLS VPNs and run a callback function with user data for each MPLS VPN.

Chapter 27: Q-in-Q Bridging

27.1 Introduction

The Q-in-Q application defines L2 VSI/VPNs (Virtual Private LAN) that are bridging domains (VSI). Each VSI performs MAC-based forwarding, SA learning, and unknown destination flooding within the VPN.

The Q-in-Q APIs aim to provide support for the IEEE 802.1ad standard. The application can be seen as an enhancement of a simpler bridge (IEEE 802.1q) that is described by Chapter 21, Ethernet Bridge.

Steps that are described by Chapter 21, Ethernet Bridge and Chapter 16, VLAN Editing Mechanism are referred only by the Q-in-Q section.

Virtual Switch (Vswitch) is of type L2VPN whenever the attached LIFs are Attachment Circuits (AC-LIFs). AC-LIFs can use the content of up to two Ethernet Tags for unique identification.

The VPN is classified to:

- Multipoint VPN – In Multipoint VPN, all the paradigms of bridging are implemented: MAC DA forwarding, MAC SA Learning, and VSI Unknown flooding. A VSI object is binding the VPN members and bridging operations can be performed on it.
- Point-to-Point P2P VPN – The Vswitch consists of only two AC-LIFs. In P2P, there is neither DA forwarding lookups, nor SA learning, nor flooding, as the mere appearance of a packet on a LIF implies its forwarding to the peer LIF. Therefore, a P2P VPN doesn't require a VSI object.

The application configuration checklist summarizes the main attributes required to configure in order for the device to meet Q-in-Q bridging requirements.

27.2 Application Configuration Checklist

- Create a MP Q-in-Q VPN
- Create Service AC-LIFs
- Associate a Service AC-LIF with a VPN
- Defining a Multicast group with in a Q-in-Q VPN
- Forwarding static configuration for MP VPN
- P2P, Cross-Connect configuration

NOTE: The above checklist, assumes the Ethernet-Bridge application is already configured.

27.3 Creating a Multipoint Q-in-Q VPN

A multipoint Q-in-Q VPN is a VSI with which Service AC-LIFs are associated. The L2VPN VSI value is usually matched with an equivalent MC-ID that is used to create the flooding groups.

The user may reserve the range 0-4K for single VLAN tag based VSIs of simple bridging.

At creation most of the VSI attributes are reset. For more information about the VSI attributes that can be modified. See Section 21.9, VSI Ethernet Bridging Properties.

27.3.1 SOC Properties

None

27.3.2 Configuration Flow

1. Create a Multipoint Q-in-Q L2VPN by calling one of the two APIs below:
   `bcm_vswitch_create(unit, vsi)`
   - `vsi` – The output allocated Vswitch VSI.

   `bcm_vswitch_create_with_id(unit, vsi)`
   - `vsi` – A user specified Vswitch VSI to allocate. The API will fail if the supplied VSI is already allocated.

   On creation, default values are set to VSI, similar to `bcm_vlan_create`, for more information, see Section 21.9, VSI Ethernet Bridging Properties.
2. Associate the VPN with multicast groups and additional properties by calling:
   `bcm_vlan_control_vlan_set`. For more information, see Section 21.9, VSI Ethernet Bridging Properties.
   Create multicast groups according to the VPN flooding groups for broadcast/unknown unicast/unknown multicast packets by calling: `bcm_multicast_create()`
   For more information, refer to the Multicast section in the *Traffic Manager Programming Guide*.

27.3.3 Shell Commands

None

27.3.4 Application Reference

`cint_vswitch_metro_mp.c`

27.4 Creating Service AC-LIFs

Service AC-LIFs are used by the L2VPN scheme. Those AC-LIFs differ from VLAN-Translation AC-LIFs (As explained by Chapter 21, Ethernet Bridge) in some of the attributes and in the configuration sequence.

Service AC-LIFs are symmetrical and therefore their configuration allocates both an InLIF and an OutLIF at the same time. Allocated InLIFs and OutLIFs share the same gport value.

Service AC-LIFs support unprotected services where a symmetric MACT learning mode applies. In this case, the InLIF gport of the incoming packet is used as the learned OutLIF (along with the destination) for packets that will Egress to the same service.

To reduce the number of allocated Service AC-LIFs, an *optimized* approach is introduced, similar to the option that exists for VLAN-Translation AC-LIFs. Such an AC-LIF can be referred to as an *Optimized Service*. This AC-LIF is characterized by a VSI assignment mode that associates the incoming packet to a VSI according to the value of its Outer-VID + a user base value (which must be a 4K modulo). This enables up to 4K services to use the same service LIF.

Protection schemes can be supported by Service AC-LIFs, both 1:1 and 1+1 schemes. AC-LIFs that are part of a protecting scheme use learn information that represents their respective protection group. For more information, see Chapter 14, Protection Switching.

At creation, an Object-ID, named VLAN-Port gport, is encoded and returned to the user to be used as a handle for various LIF based operations. The encoding of this gport is software based and used by the SDK to identify the AC LIFs. For detailed description of gport encoding, gport type and subtype, and so on, see Section 3.2, Gport Encoding of a Logical Interface.

Service AC LIF, enables the following setting for the LIF:

- Base-VSI and VSI assignment mode – Select the VSI resolution method, which is either non-optimized or optimized. For non-optimized, derive the VSI-ID directly from the AC LIF base-VSI. For the optimized service, the VSI ID is equal to base-VSI plus packet VLAN.
- Global LIF – Used as an object ID that can be used for cross-device identification both by the hardware and software BCM APIs. The Object ID is allocated from the Global InLIF and Global OutLIF ranges. For Global-LIF range information, see Section 7.3, Global LIF-ID Management.

NOTE: Global-LIF-ID must be available on both sides (In-Global-LIF and Out-Global-LIF)

- VLAN translation information – For a full description, see Chapter 16, VLAN Editing Mechanism
  Note: By default, upon AC InLIF creation, VLAN translation default properties are set to zero (VLAN Edit Profile, VID1, VID2, and QoS PCP-DEI profile).
- Ingress QoS PHB, remark, and egress QoS remark/marking – Indicate the ingress and egress QoS profiles that are used to define the PHB and remark functionality at the ingress and remarking/marking at the egress. For details, see Chapter 10, QoS.

NOTE: By default, upon creation of AC InLIF, ingress and egress QoS profile is 0. For more information, refer to Chapter 10, QoS.

- Egress QoS-Model of PHB, remark, and TTL – Indicate the type of QoS inheritance. For details, see Chapter 10, QoS.
- QOS-Parameters PCP, DEI of egress AC if QOS-Model pipe is selected.
- Learn: enable and information – Set the ingress information of destination for learning purposes. For a service AC LIF the default learning for an unprotected AC-LIF is set to the physical port and the AC-LIF. For the protected case, the FEC that represents the Protection group should be used. It is possible to enable/disable learning. Note: By default, upon creation of VLAN-translation AC-LIF, learn-enable is false.
- Stat-PP-command and Stat-PP-profile – Assign counter pointer with stat-profile to the ingress and egress AC object. For more information, see Chapter 15, PP Statistics Generation.
- Ingress and egress action-profile – Decide the trap forwarding/mirror action. For more information, see Section 12.5.6, LIF Traps. By default, the action profile is None.
- Unknown-DA: Ingress configuration. Possible to change flooding group ID offsets based on Unknown-Unicast-DA, Unknown-Multicast-DA, Unknown-Broadcast per AC-LIF.

NOTE: Only four combinations are possible for all AC LIFs. By default, upon AC-LIF creation, profile 0 is used. This values means the offset is 0 and the flooding group ID is equal to VSI (Service).

- Wide-data – Ingress general data for ACL purposes. (details can be found in Section 21.17, In-AC Wide Data).

- Additional protection fields may be set according to the required protection scheme:
  - 1+1 Protection – See Section 14.4, VLAN-Port 1+1 Protection.
  - 1:1 UC Protection – See Section 14.3.1.1, L2 Object: VLAN-Port 1:1 Unicast Protection.
  - 1:1 MC Protection – See Section 14.5, VLAN-Port 1:1 Multicast Protection.
  - Ring Protection – See Chapter 37, ITU-T G.8032 Ring Automatic Protection Switching.

After a Service AC LIF is created, some of its settings may be modified. This operation is known as a *replace*. For details about an AC *replace* operation, see Section 21.7, VLAN-Translation AC-LIF.

27.4.1 SOC Properties

None

27.4.2 Configuration Flow

1. Create a service AC LIF (optimized or non-optimized) by calling:
   `bcm_vlan_create(unit, vlan_port)`
   – `vlan_port.flags`:

Flags	Description
`BCM_VLAN_PORT_WITH_ID`	If set, provide the required allocated AC gport in `vlan_port_id`.
`BCM_VLAN_PORT_VSI_BASE_VID`	An optimized service object is created.
`BCM_VLAN_PORT_STAT_INGRESS_ENABLE`	Enable In-AC counting
`BCM_VLAN_PORT_STAT_EGRESS_ENABLE`	Enable Out-AC counting
`BCM_VLAN_PORT_INGRESS_WIDE`	The In-AC contains general (wide) data, which might be used by the ACL. (For more information, see Section 21.17, In-AC Wide Data.)
`BCM_VLAN_PORT_IVL`	If present, In-AC selects the IVL learn payload; otherwise, it selects the SVL learn payload by default. (For more information, see Section 21.19, MACT IVL.)

   – `vlan_port.criteria`: The criteria represents how the InLIF is pointed or hit

Criteria	Description
`BCM_VLAN_PORT_MATCH_NONE`	No match is set to the LIF as part of the API creation. Can be added later by `bcm_port_match_add`. Can be used by the Optimized mode.
`BCM_VLAN_PORT_MATCH_PORT`	Set the incoming-port.default-LIF to this LIF (that is, if no other match in ISEM/TCAM databases). If `match_ethertype` is nonzero, TCAM lookup of S-VLAN (=0) × C-VLAN (=0) × PCP (=0) × EtherType × VLAN-Domain instead. Can be used by the Optimized mode.
`BCM_VLAN_PORT_MATCH_PORT_VLAN`	Add ISEM lookup of S-VLAN × VLAN-Domain. If `match_ethertype` is nonzero, TCAM lookup of S-VLAN × C-VLAN (=0) × PCP (=0) × EtherType × VLAN-Domain instead
`BCM_VLAN_PORT_MATCH_PORT_VLAN_STACKED`	Add ISEM lookup of S-VLAN × C-VLAN × VLAN-Domain. If `match_ethertype` is nonzero, TCAM lookup of S-VLAN × C-VLAN × PCP (=0) × EtherType x VLAN-Domain instead
`BCM_VLAN_PORT_MATCH_PORT_CVLAN`	Add ISEM lookup of C-VLAN × VLAN-Domain. In order to match according to CVLAN , tag format of Ctag needs to be set and hit. If `match_ethertype` is nonzero, TCAM lookup of S-VLAN × C-VLAN (=0) × PCP (=0) × EtherType × VLAN-Domain instead
`BCM_VLAN_PORT_MATCH_PORT_PCP_VLAN`	Add TCAM lookup of S-VLAN × PCP × VLAN-Domain.

Criteria	Description
BCM_VLAN_PORT_MATCH_PORT_PCP_VLAN_STACKED	Add TCAM lookup of S-VLAN × C-VLAN × PCP × VLAN-Domain.
BCM_VLAN_PORT_MATCH_PORT_UNTAGGED	Add ISEM lookup of untagged packet per Incoming-Port. If match_ethertype is nonzero, TCAM lookup of S-VLAN (=0) × C-VLAN (=0) × PCP (=0) × EtherType × VLAN-Domain is performed instead.

- vlan_port.port – The physical port associated to the LIF. Derives the VLAN-domain and learning information.
- vlan_port.match_vlan – Relevant for any criteria that integrates a VLAN, such as VLAN × PORT. Represents the Outer VLAN ID for the ISEM lookup.
- vlan_port.match_inner_vlan – Relevant only for the criteria VLAN × VLAN × PORT. Represents the Inner VLAN ID for the ISEM lookup.
- vlan_port.vsi – The associated Virtual Switch instance (VSI):
  - For non-optimized services, must be different set to zero to mark the AC-LIF as a Service type. The VSI association should be performed later by calling bcm_vswitch_port_add.
  - For optimized services, must be set to a 4K modulo value that serves as the base value for the added Outer-VID.
- vlan_port.vlan_port_id – The Global LIF VLAN-Port gport (can be in or in/out according to flag BCM_VLAN_PORT_WITH_ID).

NOTE: The Global LIF of a VLAN port is limited to a 20-bit value (1M).

- Protection information:
  - vlan_port.ingress_failover_port_id – 1+1 protection, indication if primary/secondary. For more information, see Chapter 14, Protection Switching.
  - vlan_port.ingress_failover_id – 1+1 protection, ingress failover ID. The field is replaceable only if it was already set at creation.
  - vlan_port.egress_failover_id – For a description, see Chapter 14, Protection Switching.
  - vlan_port.egress_failover_port_id – For a description, see Chapter 14, Protection Switching.
  - vlan_port.failover_mc_group – Can be used only for multicast of src-sys-port learning or zero for src-sys-port learning regardless of ingress protection. For a description, see Chapter 14, Protection Switching. The field is replaceable only in case it was already set at creation.
  - vlan_port.failover_id
- vlan_port.failover_port_id – Enables modification of the learn destination to a value other than the packet's src-sys-port. The user can supply a physical port (that is, local-port, system-port, flow-id, trunk, and so on) or a protection FEC gport.

  For a physical port that was entered as local-port, system-port, or mod-port, the retrieved value for the failover_port_id field will always have a system-port gport encoding, regardless of the gport encoding that was used to set the physical port. In addition, whenever this system-port is similar to the system-port of the gport that was supplied in the vlan_port.port field, the failover_port_id will be retrieved as unset.

  For FEC gport usage, see Chapter 14, Protection Switching.
- vlan_port.group – Flush group ID that can be used by the L2 flush machine on entries that are destined to specific OutLIFs. The field is replaceable only if it was already set at creation.
- vlan_port.pkt_pri – Set PCP. This can be used if Egress QOS model is Pipe and according to Egress VLAN editing decision.
- vlan_port.pkt_cfi – Set DEI. This can be used if Egress QOS model is Pipe and according to Egress VLAN editing decision.
- vlan_port.ingress_network_group_id – Not supported for native. The field is replaceable.
- vlan_port.learn_strength – Learn strength in L2 learned entry. The field is replaceable.

2. Enable or disable learning by using `bcm_port_learn_set`:
   - `port` – VLAN-Port ID (object created from VLAN port API)
   - `flags` – `BCM_PORT_LEARN_ARL` to enable learning otherwise disable.
3. Set flooding groups offset call `bcm_port_flood_group_set(unit, port, flags, flood_groups)`.
   - `port` – VLAN-Port ID (Ingress AC LIF VLAN translation object created from VLAN port API)
   - `flags` – Not used.
   - `flood_groups` – Defined as follows:

```
typedef struct bcm_port_flood_group_s {
    bcm_gport_t unknown_unicast_group;
    bcm_gport_t unknown_multicast_group;
    bcm_gport_t broadcast_group;
} bcm_port_flood_group_t;
```

     Members in the structure are used as an offset to flooding groups for unknown unicast, unknown multicast, and broadcast packets. The value used for them is a gport that can be:
     - `BCM_GPORT_BLACK_HOLE` – Drop packets on specific InLIF or in-port. See Chapter 2, Ports and Generalized Ports, for additional configuration required for black hole.
     - `BCM_GPORT_TYPE_MCAST` – The Multicast-ID is the offset to flooding groups for Unknown packets.
     - `BCM_GPORT_INVALID` – Keep the flooding group unchanged.
4. To configure statistic command properties per AC object use: `bcm_gport_stat_set` with gport = VLAN port gport as described in Chapter 15, PP Statistics Generation.
5. Configure wide-data. For more information, see Section 21.17, In-AC Wide Data.
6. Configure an action-profile for an object by calling `bcm_rx_trap_lif_set`. See Section 12.5.6, LIF Traps.
7. Configure a QoS map profile for an object by calling `bcm_qos_port_map_set(unit, vlan_port.vlan_port_id, ingress_qos_map_id, egress_qos_map_id)`. For more information, see Chapter 10, QoS.
8. Associate a VPN to a VLAN port, see Section 27.5, Associating a Service AC-LIF with a VPN.
9. Set Protection failover ID, see Chapter 14, Protection Switching.

27.4.3 Shell Commands

None

27.4.4 Application Reference

`cint_bridge_application.c`

27.5 Associating a Service AC-LIF with a VPN

An AC-LIF becomes an integral part of a Q-in-Q L2VPN by associating it with the specific Vswitch. Service AC-LIF and Extended-LIF are the only types of AC-LIF that can be associated with an L2VPN API (`bcm_vswitch_port_*`). In other LIF types, VPN is part of the API creation.

At association, the Service AC InLIF is attributed with the vswitch VSI value and the learning is enabled according to the learning information that was set earlier by `bcm_vlan_port_create`.

Following a Service AC-LIF association, the VPN's flooding multicast group should be updated as described in Section 27.6, Defining a Multicast Group within a Q-in-Q VPN.

For Optimized Service AC-LIFs, the VPN is resolved from the packet's Outer-VID; therefore, this association step is skipped.

27.5.1 SOC Properties

None

27.5.2 Configuration Flow

1. Associate a Service AC-LIF with a VPN by calling:
   `bcm_vswitch_port_add(int unit, bcm_vlan_t vsi, bcm_gport_t port)`
   - `vsi` – The Vswitch VSI to which the AC-LIF is associated.
   - `port` – The gport of associated AC-LIF.

27.5.3 Shell Commands

None

27.5.4 Application Reference

`cint_bridge_application.c`

27.6 Defining a Multicast Group within a Q-in-Q VPN

An L2VPN flooding group should contain the participating OutLIFs. As the multicast group is created, the OutLIFs that were associated with the Vswitch may be added to the Multicast group using a utility function that retrieves the CUD from the Service AC-LIF gport.

27.6.1 SOC Properties

None

27.6.2 Configuration Flow

1. Open a TM multicast group with the Vswitch VSI in ingress and/or fabric and egress. Call:

   `bcm_multicast_create(unit, flags, *group)`

   `group` – Allocated multicast group ID

   See more info in the multicast section in Traffic Management document.

2. Associate any relevant MAC DA that is not added by learning with the TM multicast group by calling:

   `bcm_l2_addr_add(unit, l2addr)`

   – `l2addr` – L2 address entry structure. See Section 21.14, L2 MACT Forwarding.
   - `flags |= BCM_L2_MCAST` – Indicate a MC group destination
   - `flags |= BCM_L2_STATIC` – Indicate a static MACT entry
   - `l2mc_group` – The destination multicast group `(mc_group)`, supplied instead of any other destination field (`port, encap_id` etc).

3. For each associated Service AC-LIF, retrieve the CUD value:

   `bcm_multicast_vlan_encap_get(unit, group, port, vlan_port_id, *encap_id)`

   – `vlan_port_id` – Service AC-LIFs gport

   – `encap_id` – Retrieved CUD value that represents the global OutLIF

4. Add the AC-LIFs CUD to the multicast group:

   `bcm_multicast_add(unit, group, flags, nof_replications, rep_array)`

   – `group` – TM multicast allocated group (`group`)

   – `rep_array, nof_replications` – Update the replications list with the CUDs (`encap_id`) and the number of the supplied replications in the array. For more info refer to the Multicast section of the *Traffic Manager Programming Guide*.

27.6.3 Shell Commands

None

27.6.4 Application Reference

`cint_sand_l2_multicast.c`

27.7 Split Horizon Filtering

Split Horizon filtering for a Vswitch can be achieved by setting the Orientation group for associated AC and PWE LIFs. For details about setting the filter, see Chapter 20, L2 Generalized Bridging Model.

27.7.1 SOC Properties

- `in_lif_profile_allocate_orientaion`

27.7.2 Configuration Flow

1. Set an Orientation group for both the InLIF and OutLIF of Service AC at creation:

   `bcm_vlan_port_create(unit, vlan_port)`

   In the vlan_port structure, there are the following Orientation group fields:

   – `ingress_network_group_id` – Ingress Orientation group in the range of the SOC Property `in_lif_profile_allocate_orientaion`.

   – `egress_network_group_id` – Egress Orientation group

   For other parameters, see Section 27.4, Creating Service AC-LIFs.

2. Optionally update the Orientation group for a Service AC-LIF by calling the previous API with modified Orientation group values and the following flags:

   – `BCM_VLAN_PORT_WITH_ID` – Indicates the AC LIF gport is supplied by the user.

   – `BCM_VLAN_PORT_REPLACE` – Indicates that the command modifies an allocated object instead of creating one.

3. Define Orientation-based filtering by calling:

   `bcm_switch_network_group_config_set(unit,source_network_group_id, config)`

   For details, see Chapter 20, L2 Generalized Bridging Model.

27.7.3 Shell Commands

None

27.8 Forwarding Static Configuration for MP VPN

MACT entries can be added statically and can also be added dynamically as a result of a learning process. When referring to MACT, the SDK can configure the MAC table. For more information, see Section 21.14, L2 MACT Forwarding.

A Service AC may be used both as an input that is later to be resolved to a destination, and as an additional OutLIF information to a specified destination.

27.8.1 SOC Properties

None

27.8.2 Configuration Flow

Add a static MAC entry by calling: `bcm_l2_addr_add(unit, l2addr)`.

- `l2addr.port` – Can be configured to a physical port (such as a system-port, local-port or flow-id) or a Service-AC gport.
- `l2addr.encap_id` – Service Out-AC can be used when the port is a physical-port type and a logical-interface pointer is required too. This is useful for multidevice cases (when the logical-gport is not exist in the local device).

For more information regarding the API fields, see Section 21.14, L2 MACT Forwarding.

27.8.3 Shell Commands

See Section 21.14, L2 MACT Forwarding.

27.8.4 Application Reference

`cint_l2_basic_bridge_with_vlan_editing.c`

27.9 P2P Service

A P2P relation can be configured between two AC or PWE objects. This type of service encompasses the forwarding information with the InLIF, skipping the need of a MACT lookup.Focusing on an AC object, it can represent either an unprotected AC-LIF or a protection pair. The association can be configured as unidirectional or bidirectional.

At InLIF creation for AC LIFs, the AC InLIF is created as a multipoint (MP) LIF. When cross-connecting it to the destination gport, the InLIF is changed to a P2P InLIF that stores the forwarding information. When disassociating the cross-connection, the InLIF is changed back to its default as an MP-LIF.

27.9.1 SOC Properties

None

27.9.2 Configuration Flow

- Create an AC LIF: `bcm_vlan_port_create(unit, vlan_port).`

NOTE: By default, the InLIF is created as an MP-LIF.

For other fields and additional flags refer tho the example in Section 27.4, Creating Service AC-LIFs.

For a protection pair creation, see Chapter 14, Protection Switching.

- In case of a bidirectional association between two ACs, the paired AC-LIF should be created as the first.
- Associate the P2P AC-LIF with a destination gport by calling `bcm_vswitch_cross_connect_add(unit, *gports).`

NOTE: The InLIF type is modified from an MP-LIF to a P2P-LIF.

For more information, see Section 20.4.2, Cross Connect.

27.9.3 Shell Commands

None

27.9.4 Application Reference

`cint_vswitch_cross_connect_p2p.c`

27.10 API Descriptions

27.10.1 Multipoint Q-in-Q VPN

API Name	Highlights
`bcm_vswitch_create(int unit, bcm_vlan_t *vsi)`	Create a virtual switch with no specific VSI ID
`bcm_vswitch_create_with_id(int unit, bcm_vlan_t vsi)`	Create a virtual switch with a specified VSI ID
`bcm_vswitch_destroy(int unit, bcm_vlan_t vsi)`	Destroy a virtual switch
`bcm_vlan_control_vlan_set(int unit, bcm_vlan_t vlan, bcm_vlan_control_vlan_t control)`	Set additional VSI properties
`bcm_vlan_control_vlan_get(int unit, bcm_vlan_t vlan, bcm_vlan_control_vlan_t *control)`	Get additional VSI properties

27.10.2 Service AC-LIFs

API Name	Highlights
`bcm_vlan_port_create (int unit, bcm_vlan_port_t * vlan_port)`	Create a L2 logical port
`bcm_vlan_port_destroy (int unit, bcm_gport_t gport)`	Destroy a L2 logical port
`bcm_vlan_port_find (int unit, bcm_vlan_port_t * vlan_port)`	Get/find a layer 2 logical port given the GPORT id or match criteria
`bcm_vlan_port_traverse (int unit, bcm_vlan_port_traverse_info_t *additional_info, bcm_vlan_port_traverse_cb trav_fn, void *user_data)`	Traverse VLAN-Port objects according to specified rules
`bcm_port_learn_set(int unit, bcm_port_t port, uint32 flags)`	Enable learn per port

API Name	Highlights
`bcm_port_learn_get(int unit, bcm_port_t port, uint32 *flags)`	Get learn enabled/disabled per port
`bcm_port_flood_group_set(int unit, bcm_gport_t port, uint32 flags, bcm_port_flood_group_t *flood_groups)`	Set the flood groups for for port or InLIF
`bcm_port_flood_group_get(int unit, bcm_gport_t port, uint32 flags, bcm_port_flood_group_t *flood_groups)`	Get the flood groups for for port or InLIF

27.10.3 Associating a Service AC-LIF with a VPN

API Name	Highlights
`bcm_vswtich_port_add (int unit, bcm_vlan_t vsi, bcm_gport_t port)`	Associate a Service AC-LIF with a virtual switch VSI
`bcm_vswitch_port_delete (int unit, bcm_vlan_t vsi, bcm_gport_t port)`	Deletes an association of a Service AC-LIF to a virtual switch VSI
`bcm_vswitch_port_delete_all (int unit, bcm_vlan_t vsi)`	Deletes all association of LIFs to a virtual switch VSI **NOTE:** Zeroed VSI disassociates all LIFs
`bcm_vswitch_port_get (int unit, bcm_gport_t port, bcm_vlan_t *vsi)`	Retrieves whether an AC-LIF is associated with a virtual switch.

27.10.4 Multicast Group within a Q-in-Q VPN

API Name	Highlights
`bcm_multicast_vlan_encap_get(int unit,bcm_multicast_t group, bcm_gport_t port, bcm_gport_t vlan_port_id, bcm_if_t * encap_id)`	Retrieve MC-ID from VLAN-Port information

27.10.5 Forwarding Static Configuration for MP VPN

API Name	Highlights
`bcm_l2_addr_add (int unit, bcm_l2_addr_t *l2addr)`	Add a MAC Table entry
`bcm_l2_addr_get(int unit, bcm_mac_t mac_addr, bcm_vlan_t vid, bcm_l2_addr_t *l2addr)`	Retrieve a MAC Table entry
`bcm_l2_addr_delete (int unit, bcm_mac_t mac, bcm_vlan_t vid)`	Delete a MAC Table entry

27.10.6 P2P Service

API Name	Highlights
`bcm_vlan_port_create (int unit, bcm_vlan_port_t * vlan_port)`	Create a L2 logical port
`bcm_vlan_port_destroy (int unit, bcm_gport_t gport)`	Destroy an L2 logical port
`bcm_vlan_port_find (int unit, bcm_vlan_port_t * vlan_port)`	Get/find a layer 2 logical port
`bcm_vswitch_cross_connect_add (int unit, bcm_vswitch_cross_connect_t * gports)`	Connect two logical ports in a P2P service **NOTE:** For an AC LIF, this API may change an InLIF format from an MP-LIF to a P2P-LIF.
`bcm_vswitch_cross_connect_get (int unit, bcm_vswitch_cross_connect_t * gports)`	For a given P2P gport, return its P2P peer
`bcm_vswitch_cross_connect_delete (int unit, bcm_vswitch_cross_connect_t * gports)`	For a given P2P gport, delete the connection to its P2P peer **NOTE:** For an AC LIF, this API may change an InLIF format from a P2P-LIF to an MP-LIF.
`bcm_vswitch_cross_connect_delete_all (int unit)`	Delete all the connections between any P2P peers **NOTE:** For an AC LIF, this API may change an InLIF format from a P2P-LIF to an MP-LIF.
`bcm_vswitch_cross_connect_traverse (int unit, bcm_vswitch_cross_connect_traverse_cb cb, void *user_data)`	Traverse all the cross-connect connections between any P2P peers

Chapter 28: IP Tunnel v4 Termination

28.1 Introduction

IPv4 termination application defines the means by which packets are switched into and out of an IP domain. This section provides information on how to configure IPV4 L3 tunnel termination and what actions are taken prior to the forwarding on the main header (MPLS/IP). The section refers mainly to how the Tunnel termination stages are configured. Also the API calls for defining IPV4 ingress tunnel are described. For L2 Tunnel termination cases, see Chapter 32, VXLAN IP Overlay.

It is recommended to read this section after the following sections:

- Chapter 21, Ethernet Bridge and Chapter 22, IP Router to get an overview of the basic L2 and L3 objects.
- Section 22.6, ETH-RIF (L3 VSI Interface) and VRF Definition. This section provides information about configurations that are related to next hop information, namely FEC and Egress-ARP. Also, it provides information relevant for the MyMAc operation which is a prerequisite for IP tunnel forwarding and termination.

28.2 Application Configuration Checklist

The IP tunnel v4 termination configuration checklist is as follows:

- Set L3 port properties – For more information, see Section 22.3, Port Routing Properties.
- Set ETH-RIF properties and create MyMac per ETH-RIF. For more information, see Section 22.6, ETH-RIF (L3 VSI Interface) and VRF Definition.
- Ingress IP-tunnel termination object.

28.3 Ingress L3 IP-Tunnel IPv4 Termination Object

L3 IP-Tunnel IPv4 termination with or without GRE header is done as part of the Tunnel termination processing at the device. Tunnel Termination procedure is done right after the first ETH header termination (if My-MAC is set).

The following L3 IP-Tunnel IPv4 types are supported:

- GreAnyIn4 – IP header with GRE header. Following GRE types are supported:
  - Gre4oIPv4 header with payload MPLS/IPv4/IPv6.
  - Gre8oIPv4 with entropy key and payload MPLS/IPv4/IPv6.
- UDP – IP header with UDP header. Two different flavors are supported:
  - UDP next protocol is not VXLAN.
  - UDP next protocol is VXLAN (and VXLAN-GPE). For such a support see Chapter 32, VXLAN IP Overlay.
- IpAnyIn4 – IP header termination is done without any additional headers (GRE/UDP)

The following termination lookup keys are supported by the SDK:

- <SIP, DIP, VRF, Tunnel-Type> in ISEM, if both SIP and DIP are configured as exact match
- <DIP, VRF, Tunnel-Type> in ISEM, if only DIP is configured as exact match, the SIP is fully masked.
- <SIP, DIP, VRF, Tunnel type> in TCAM, if SIP and DIP are partially masked.

If the tunnel termination lookup is successful, the result is an IP-Tunnel termination LIF object. According to LIF properties and the next-header, the routing-enabler procedure decides if tunnel termination is done successful or a trap is invoked.

NOTE: A routing-enabler procedure exist for IP and MPLS tunnels. It is important to correctly configure the tunnel next protocols enabler to allow a successful termination.

Tunnel termination will not occur when the layer above the IPv4 or IPv6 tunnel is one of the following:

- TCP
- UDP
- GRE with keep alive
- ICMP
- IGMP
- Unknown

This list does not include SRv6 cases when the Segment Left field is 0. A new control is provided to perform (or avoid performing) the tunnel termination procedure in this case.

The IP-Tunnel termination object is considered to be an L3 incoming interface for the next termination/forwarding header.

NOTE: An IP Tunnel termination object is a RIF-L3 interface by itself and all RIF-L3 interface properties are part of the object.

The following settings are possible to configure per IP-Tunnel termination object:

- Tunnel-Type – Decide the tunnel-type of the object. This cannot be changed once the object is created.
- QoS model – Allow pipe or uniform DSCP assignment.
- ECN – Tunnel allows ECN to be eligible or not. Valid for VXLAN GPE only.
- TTL model – Allow pipe or uniform TTL inheritance.

- Termination on first or second pass – This is widely used in BUD node configuration - drop and continue flow, in which it is performed simultaneous forwarding of copies of a received multicast packet on the multicast tree (continue), and off to a locally attached network (drop). The drop of packets to the local network requires termination of the IP tunnel, which is executed on the second pass of the packet through the ingress pipe. The lookup keys are the same as in first pass tunnel termination.
- L3 interface properties: following list of fields, For more information on each field, see Section 22.6, ETH-RIF (L3 VSI Interface) and VRF Definition.
  - VRF
  - Routing enabled: Enable or disable routing per protocol, including IPv4 and IPv6 unicast or multicast, MPLS, and native Ethernet.
  - Unicast RPF
  - Enable/disable public routing

28.3.1 VRF Update Before Tunnel Termination

VRF can be updated before tunnel termination lookups using a PBR-based TCAM lookup. This lookup can be enabled or disabled using `bcm_switch_control_set() type=bcmSwitchL3VpbrTunnel`. When enabled, the TCAM lookup will not be used for termination and instead it will be used for VRF update.

VRF update before tunnel termination is supported only for IPv4 tunnels.

28.3.2 SOC Properties

None

28.3.3 Configuration Flow

1. To create an ingress IP tunnel, call:
   `bcm_tunnel_terminator_create(unit, tunnel)`
   – `tunnel.flags` – The following flags are supported:

Supported Flags	Description
`BCM_TUNNEL_TERM_TUNNEL_WITH_ID`	Create a tunnel terminator with Global LIF ID passed by tunnel->tunnel_id
`BCM_TUNNEL_REPLACE`	Replace an already allocated tunnel terminator entry. The flag can be used only with the *`BCM_TUNNEL_WITH_ID`* flag and a valid tunnel_id.
`BCM_TUNNEL_TERM_BUD`	Create the tunnel terminator lookup key in second pass termination tables. This feature is useful for BUD Node processing.
`BCM_TUNNEL_TERM_STAT_ENABLE`	Enable statistics counting for the tunnel termination LIF object.

   – `tunnel.vrf`: Configure vrf value, this vrf is resolved after ETH termination or previous IP/MPLS-Tunnel and is part of the three possible tunnel terminator key combinations lookup keys.
   – `tunnel.priority`: If the tunnel terminator entry is written in the TCAM, priority can specified. If this field is omitted, the default priority 0 is given.
   – `tunnel.sip`: Configure tunnel SIP as part of the lookup key.
   – `tunnel.dip`: Configure tunnel DIP as part of the lookup key.
   – `tunnel.sip_mask`: Configure tunnel SIP mask.
   – `tunnel.dip_mask`: Configure tunnel DIP mask.
   – `tunnel.ingress_qos_model`: Configure tunnel QoS-model and TTL-model. For more information, see Section 10.3, Ingress PHB/Remarking.

- `tunnel.ingress_qos_model.ingress_ecn`: Configure a tunnel to be ECN eligible or ineligible. Available for VXLAN or VXLAN GPE tunnel only. For more information, see Section 10.6, Explicit Congestion Notification.
- `tunnel.default_vrf`: Configure the default VRF. Use this option if there is no existing mapping between the VNI and VRF. Use this for a VXLAN-GPE tunnel with IPv4 or IPv6 forwarding.
- `tunnel.type`: Configure the tunnel type of the tunnel to be terminated. Supported IPv4 tunnel types for termination are described in the following table.

Supported Types	Description
`bcmTunnelTypeIpAnyIn4`	Configures all IPv4 tunnels, the next-layer support is decided according to routing enablers. The mention of *Any* in the tunnel type is different than the equivalent IPv6 tunnel type (see bcmTunnelTypeIpAnyIn6). IPv4 tunnels means that there are no additional headers to the IPv4 tunnel (such as GRE, UDP, VXLAN, and so on).
`bcmTunnelTypeGreAnyIn4`	Configures all GreoIpv4 tunnels, the next-layer support is decided according to routing enablers.
`bcmTunnelTypeUdp`	Configures UDPoIPv4 tunnels. Source and destination port are not required for configuration, they are resolved according to the routing enablers.

2. Additional IP-Tunnel properties can be set (for example, Routing enablers, VRF) by calling: `bcm_l3_ingress_create (unit, ingr_intf, intf_id)` API with the following parameters:
   - `ingr_intf`: See detailed information in Section 22.6, ETH-RIF (L3 VSI Interface) and VRF Definition.
   - `intf_id`: convert the created tunnel terminator ID to L3 interface LIF ID type with the macro: `BCM_GPORT_TUNNEL_TO_L3_ITF_LIF(intf_id, tunnel_term.tunnel_id);`
3. If statistics are enabled, configure their settings by calling the API `bcm_gport_stat_set(unit, gport, core_id, engine_source, stat_info)` where `gport` is the Tunnel termination object ID. For more information about the API see Chapter 15, PP Statistics Generation.

General Configuration

To enable or disable the tunnel termination process for a specific layer above the IPv4 or IPv6 tunnels, call `bcm_switch_control_set (unit, bcmSwitchTunnelRouteDisable, arg)`

For the `arg` parameter, the following flag or combination of flags per layer above the tunnel can be used:

- BCM_SWITCH_TUNNEL_ROUTE_DISABLE_NONE
- BCM_SWITCH_TUNNEL_ROUTE_DISABLE_TCP
- BCM_SWITCH_TUNNEL_ROUTE_DISABLE_UDP
- BCM_SWITCH_TUNNEL_ROUTE_DISABLE_ICMP
- BCM_SWITCH_TUNNEL_ROUTE_DISABLE_IGMP
- BCM_SWITCH_TUNNEL_ROUTE_DISABLE_UNKNOWN
- BCM_SWITCH_TUNNEL_ROUTE_DISABLE_GRE_KEEP_ALIVE

28.3.4 Shell Commands

None

28.3.5 Application Reference

TBD

Additional tunnel termination scenarios, including tunnel termination priorities, VXLAN-GPE termination, and tunnel termination with a UC/MC RPF check:

- **Type**: CINT reference
- **Path**: `$SDK/src/examples/dnx/tunnel/cint_dnx_ip_tunnel_termination.c`

28.4 API Descriptions

API Name	Highlights
`bcm_tunnel_terminator_create()`	Create Ingress tunnel and set its properties.
`bcm_tunnel_terminator_delete()`	Destroy Ingress tunnel given LIF-ID.
`bcm_tunnel_terminator_get()`	Retrieve Ingress tunnel properties given tunnel LIF-ID.
`bcm_tunnel_terminator_traverse()`	Traverse over all Ingress tunnel LIF objects.

Chapter 29: IP Tunnel v4 Encapsulation

29.1 Introduction

The IPv4 encapsulation application defines how packets are switched into an IP domain. This chapter provides information on how to configure IPv4 egress tunnel encapsulation and what actions occur after the forwarding on the main header (MPLS/IP). This chapter also describes the API calls for defining IPv4 egress tunnels. For L2/L3 tunnel encapsulation by VXLAN, see Chapter 32, VXLAN IP Overlay. For IPv6 egress tunnels, see Chapter 31, IP Tunnel v6 Encapsulation

To gain the most value from this chapter, read the following sections first:

- Chapter 21, Ethernet Bridge and Chapter 22, IP Router to get an overview of the basic L2 and L3 objects.
- Section 22.6, ETH-RIF (L3 VSI Interface) and VRF Definition. This section provides information about configurations that are related to next hop information, namely FEC and Egress-ARP. Also, it provides information relevant for the MyMAc operation which is a prerequisite for IP tunnel forwarding and encapsulation.

29.2 Application Configuration Checklist

The IP Tunnel v4 Encapsulation configuration checklist includes the following:

- Set L3 port properties – For more information, see Section 22.3, Port Routing Properties.
- Set ETH-RIF properties and create MyMac per ETH-RIF. For more information, see Section 22.6, ETH-RIF (L3 VSI Interface) and VRF Definition.
- Create Egress object: Egress-ARP. For more information, see Section 22.7, L3 Egress Object: Egress-ARP (Next-Hop).
- Egress IP-tunnel encapsulation object.

29.3 Egress L3 IP-Tunnel IPv4 Encapsulation Object

IP tunnel encapsulation with or without additional headers is done at the Encap stages in the device. The procedure is done according to the tunnel LIF object retrieved from the EEDB. The LIF object may be pointed by the ingress databases (such as forwarding databases, FEC/ECMP, and so on) or from egress databases (such as Egress-ACL) or from the EEDB itself by a previous MPLS/IP tunnel. The next-hop information (ARP) is usually pointed by the tunnel LIF object.

The following settings are available for configuring per-egress IP tunnels:

- Source and Destination Tunnel IP addresses – Used for routing inside the tunnel.
- Tunnel-Type – Specify the type of the tunnel to be encapsulated (IPv4/6, VXLAN, UDP, and so on) For a summary of all supported L3 IPv4 tunnel types, see Table 92, L3 IPv4 Tunnel Types.
- QoS properties – Two modes are supported: Pipe and Uniform inheritance.
- Encap Phase – Specify in which encapsulation EEDB phase the tunnel will be encapsulated. This is required when the L3 IPv4 tunnel is part of multiple tunnels (of IP/MPLS) encapsulation.
- Statistics counting – Enable statistics counting on the egress tunnel LIF ID. For more information, see Chapter 15, PP Statistics Generation
- MTU – Provide the MTU value for Egress MTU filtering. For more information, see Chapter 12, Traps.

  Note that by default, upon creation of tunnel LIF, MTU filter is disabled.
- Action profile – Specify the action occurs on the packet after it qualifies on an MTU trap. For more information, see Chapter 12, Traps.
- Remark profile – Set the remark profile associated with the tunnel LIF. For more information, see Chapter 10, QoS.
- ECN – Tunnel allows ECN to be eligible or not. See more information in Chapter 10, QoS. Valid for VXLAN GPE only.

29.3.1 SOC Properties

None

29.3.2 Configuration Flow

1. To create an egress IP tunnel, call: `bcm_tunnel_initiator_create(unit, &intf, &tunnel)`
   - intf

     `l3a_intf_id` – Global interface id of the created tunnel.
   - `tunnel.flags`:

Supported Flags	Description
`BCM_TUNNEL_WITH_ID`	Create a tunnel initiator with LIF ID passed by *`tunnel.tunnel_id`* or *`intf.l3a_intf_id`*. Both fields will be updated with the created tunnel id.
`BCM_TUNNEL_REPLACE`	Replace an already allocated tunnel initiator entry. The flag can be used only with the *`BCM_TUNNEL_WITH_ID`* flag and a valid tunnel ID.
`BCM_TUNNEL_INIT_GRE_KEY_USE_LB`	Create GRE8 tunnel with load balancing key. The flag is used only with a flag used with *`bcmTunnelTypeGreAnyIn4`* tunnel type.
`BCM_TUNNEL_INIT_IPV4_SET_DF`	Enable IPV4 de-fragmentation bit.
`BCM_TUNNEL_INIT_STAT_ENABLE`	Enable statistics counting for the tunnel initiator LIF object. Note: It is not possible to move between STAT disable/enable upon replace.

   - `tunnel.ttl` – Specify the value of the TTL field on the encapsulated tunnel in pipe model of TTL inheritance.
   - `tunnel.dip` – Configure tunnel DIP to be encapsulated.
   - `tunnel.sip` – Configure tunnel SIP to be encapsulated.
   - `tunnel.dscp_sel` – Configure assignment model of dscp. Two options are supported:
     - `bcmTunnelDscpAssign` – Configure pipe model of DSCP assignment. the DSCP value will be taken as tunnel LIF property.
     - `bcmTunnelDscpMap` – Configure uniform model of DSCP inheritance. The DSCP value will be taken from the forwarding header.
   - `tunnel.dscp` – DSCP value to be used for the pipe mode of inheritance.
   - `tunnel.egress_qos_model.egress_ecn`: Configure a tunnel to be ECN eligible or not. Available for VXLAN or VXLAN GPE tunnel only. For more information, see Section 10.11, Egress MPLS PHP QoS.
   - `tunnel.egress_qos_model.egress_ttl` – Configure the tunnel TTL model (`bcmQosEgressModelUniform` for Uniform or `bcmQosEgressModelPipeNextNameSpace` for Pipe)
   - `tunnel.tunnel_id` – Configure tunnel initiator global LIF ID, when creating the tunnel with `BCM_TUNNEL_WITH_ID flag.`
   - `tunnel.l3_intf_id` – If valid, pointer to the next hop information in EEDB. This ID is received from `bcm_l3_egress_create()` API (egress side). For more information on ARP entries, see Chapter 22, IP Router. If no connection to another EEDB entry exists, set to `BCM_IF_INVALID`.

NOTE: By default, the value is set to 0, which means reserve the next EEDB pointer in the entry for future use (the value can be changed upon replace).

   - `tunnel.encap_access` – Specify from which encapsulation stage the tunnel encapsulation procedure will start.

NOTE: Upon `bcm_tunnel_initiator_get` for LIF at EEDB phase 3, the returned `encap_access` parameter returns `bcmEncapAccessInvalid` (this is the default access phase)

Supported Types	Description
`bcmEncapAccessInvalid`	Default (same as Tunnel1)
`bcmEncapAccessTunnel1`	Access local OutLIF at EEDB phase 3
`bcmEncapAccessTunnel2`	Access local OutLIF at EEDB phase 4

Supported Types	Description
bcmEncapAccessTunnel3	Access local OutLIF at EEDB phase 5
bcmEncapAccessTunnel4	Access local OutLIF at EEDB phase 6

– tunnel.type: Configure the tunnel type of the tunnel to be encapsulated. Supported IPv4 tunnel types for encapsulation are:

Table 92: L3 IPv4 Tunnel Types

Supported Types	Description
bcmTunnelTypeIpAnyIn4	Used for configuration of all IPv4 tunnels.
bcmTunnelTypeGreAnyIn4	Used for for configuration of all GreoIpv4 tunnels. The GRE8 tunnel encapsulation is configured with *BCM_TUNNEL_INIT_GRE_KEY_USE_LB flag*
bcmTunnelTypeUdp	Used for configuration of UDPoIPv4 tunnels. Source and destination port are not required for configuration, they are resolved according to the forwarding layer type.

For more detailed information, see Section 7.4, EEDB Management.

2. If statistics are enabled, configure their settings by calling the API bcm_gport_stat_set(unit, gport, core_id, engine_source, stat_info) where gport is the egress tunnel LIF ID. For more information about the API see Chapter 15, PP Statistics Generation.
3. If MTU trap filtering is required on the tunnel LIF ID, associate it with MTU by: bcm_rx_mtu_set(unit, mtu_config), where mtu_config is configured as following:
   – mtu_config.flags = BCM_RX_MTU_LIF;
   – mtu_config.gport = tunnel.tunnel_id;

   For other fields and detailed information on MTU configuration and its full sequence, see Chapter 12, Traps.
4. To configure egress remark profile associated with the tunnel LIF ID, call the APIs bcm_qos_map_create(unit, flags, egr_map), where flags are configured – BCM_QOS_MAP_EGRESS, BCM_QOS_MAP_REMARK. To associate the tunnel LIF ID with the remark profile call bcm_qos_port_map_set(unit, port, ing_map, egr_map), where port is tunnel.tunnel_id. Ingress map is not required and can be left 0. For more detailed information, see Chapter 10, QoS.

29.3.3 Shell Commands

None

29.3.4 Application Reference

Example of basic encapsulation API usage for GRE4oIPv4 tunnel

- **Type:** CINT reference
- **Path:** `$SDK/src/examples/sand/cint_ip_tunnel_encap_basic.c`
  `$SDK/src/examples/dnx/tunnel/cint_dnx_ip_tunnel_encapsulation.c`

29.4 UDP Destination Port

Users can update the UDP destination port. The UDP destination port indicates the next UDP protocol. For a UDP tunnel, the next protocol is the protocol of the forwarding header.

If not configured, a default UDP destination port applies. The default port numbers are as follows:

- IPv4 – 26117
- IPv6 – 26133
- MPLS – 6653

29.4.1 SOC Properties

None

29.4.2 Configuration Flow

To configure the UDP destination port, call API `bcm_switch_control_set(unit, type, arg)`, where

- `type`:
  - `bcmSwitchUdpTunnelIPv4DstPort` for a UDP tunnel with IPv4 as forwarding header
  - `bcmSwitchUdpTunnelIPv6DstPort` for a UDP tunnel with IPv6 as forwarding header
  - `bcmSwitchUdpTunnelMplsDstPort` for a UDP tunnel with MPLS as forwarding
- `arg` – New value for the UDP destination port

29.4.3 Shell Commands

None

29.4.4 Application Reference

None

29.5 API Descriptions

29.5.1 Egress L3 IP-Tunnel IPv4 Encapsulation Object

API Name	Highlights
`bcm_tunnel_initiator_create()`	Create Egress tunnel and set its properties.
`bcm_tunnel_initiator_clear()`	Destroy Egress tunnel initiator given LIF-ID.
`bcm_tunnel_initiator_get()`	Retrieve Egress tunnel properties given Tunnel LIF-ID.
`bcm_tunnel_initiator_traverse()`	Traverse over all Egress tunnel LIF objects.

29.5.2 UDP Destination Port

API Name	Highlights
`bcm_switch_control_set()`	Update UDP destination port for UDP tunnel

Chapter 30: IP Tunnel v6 Termination

30.1 Introduction

IPv6 termination application defines the means by which packets are switched into out of an IPv6 domain. This section provides information on how to configure IPV6 L3 tunnel termination and what actions are taken prior to the forwarding on the main header (MPLS/IP). The section refers mainly to how the Tunnel termination stages are configured. Also the API calls for defining IPV6 ingress tunnel are described. For L2 v6 Tunnel termination cases, see Chapter 32, VXLAN IP Overlay.

For further information it is recommended to read this section after the following sections:

- Chapter 21, Ethernet Bridge and Chapter 22, IP Router to get an overview of the basic L2 and L3 objects.
- Section 22.6, ETH-RIF (L3 VSI Interface) and VRF Definition. This section provides information about configurations that are related to next hop information, namely FEC and Egress-ARP. Also, it provides information relevant for the MyMAc operation which is a prerequisite for IP tunnel forwarding and termination.

30.2 Application Configuration Checklist

- Set L3 port properties. For more information, seeSection 22.3, Port Routing Properties.
- Set ETH-RIF properties and create MyMac per ETH-RIF. For more information, see Section 22.6, ETH-RIF (L3 VSI Interface) and VRF Definition.
- Ingress IP-tunnel termination object.

30.3 Ingress L3 IP-Tunnel v6 Termination Object

L3 IP-Tunnel v6 termination with or without GRE header is done as part of the Tunnel termination processing at the device. Tunnel Termination procedure is done right after the first ETH header termination (if My-MAC is set).

The following L3 IP-Tunnel v6 types are supported:

- GreAnyIn6 – IP header with GRE header. Following GRE types are supported:
  - Gre4oIPv6 header with payload MPLS/IPv4/IPv6.
  - Gre8oIPv6 with entropy key and payload MPLS/IPv4/IPv6.
- UDP – IP header with UDP header. Two different flavors are supported:
  - UDP next protocol is not VXLAN.
  - UDP next protocol is VXLAN (and VXLAN-GPE). For such support, see Chapter 32, VXLAN IP Overlay.
- IpAnyIn6 – IP header termination is done without needing any additional headers (GRE/UDP).

The SDK supports two types of IPv6 Tunnel Termination – Point-to-Point, referred as v6-P2P and Multipoint, referred as v6-MP. Basically, v6-P2P identifies the tunnel by source and destination while v6-MP identify tunnel by destination only.

NOTE: The term *P2P* can be a little misleading because it is used in other protocols as a cross-connect connection between incoming-interface and outgoing-interface. This is not the case here. Here we refer to v6-P2P as for the identification method.

v6-P2P requires identifying source and destination as part of the lookup key. Since the lookup key is too large to fit into one table, the lookup is split into two steps (cascade-lookup):

1. Lookup {VRF, v6-DIP} in TCAM and the result is Intermediate object my-VTEP-Index
2. Lookup {my-VTEP-Index, v6-SIP} in SEM and the result is the LIF-object for IPv6 Tunnel termination process.

The lookup key for the v6-MP tunnel termination is built as follows: Lookup {VRF, v6-DIP} in TCAM and the result is the LIF-object for IPv6 tunnel termination process.

A certain packet is v6-P2P or v6-MP process according to the SIP. If a SIP match exists, v6-P2P processing occurs; otherwise v6-MP processing occurs.

The following table summarizes the lookups performed for the IPv6 Tunnel-Termination.

IPv6 Type	Lookup Key	Result
v6-P2P	{VRF, DIP}	my-VTEP-Index
	{my-VTEP-Index, SIP}	IPv6-Tunnel-LIF
v6-MP	{VRF, DIP}	IPv6-Tunnel-LIF

In both cases, when the tunnel termination lookup is successful, the result is an IP-Tunnel termination LIF object.

Since both Ingress L3 IP-Tunnel v4 and v6 termination objects have many attributes in common, this document provides only the differences for v6 as compared to v4. For full information on v4, see Section 28.3, Ingress L3 IP-Tunnel IPv4 Termination Object.

NOTE: If the same <VRF, DIP> serves for both MP-defined tunnel and Intermediate-object for P2P tunnel (i.e. <VRF, DIP> used for both MP and P2P defined tunnels) then a different sequence of APIs are introduced.

Call `bcm_tunnel_terminator_create` twice: Once for MP defined tunnel and once for P2P defined tunnel. More details are provided in the following sections.

30.3.1 SOC Properties

None

30.3.2 Configuration Flow

30.3.2.1 IPv6-MP Tunnel Terminator Object

To create an ingress IPv6-MP tunnel terminator (no SIP hit required), call `bcm_tunnel_terminator_create(unit, tunnel)`

- `tunnel.dip6`: Configure tunnel DIP as part of the lookup key.
- `tunnel.dip6_mask`: Configure tunnel DIP mask.
- `tunnel.priority`: Priority of the TCAM entry, if this field is omitted, the default priority 0 is given.
- `tunnel.type`: Configure the tunnel type of the tunnel to be terminated. Supported IPv6 tunnel types for termination are:

Supported Types	Description
`bcmTunnelTypeIpAnyIn6`	Used for configuration of all IPv6 tunnels (including all GreoIPv6 tunnels), UDPoIPv6 tunnels, and VXLAN-GPEoIPv6 tunnels, the next-layer support is decided according to routing enablers.
`bcmTunnelTypeGreAnyIn6`	Used for configuration of all GreoIpv6 tunnels, the next-layer support is decided according to routing enablers. **NOTE:** GreoIpv6 tunnels should be configured with higher priority than IpAnyIp6 tunnels.
`bcmTunnelTypeUdp6`	Used for configuration of UDPoIPv6 tunnels. Source and destination port are not required for configuration, they are resolved according to the routing enablers.

- All other fields and additional properties APIs are the same as IPv4 object.

30.3.2.2 IPv6-P2P Tunnel Terminator Object

Configuring an ingress IPv6 P2P tunnel terminator (SIP hit required) requires usage of 2 APIs:

1. Create intermediate object for {DIP, VRF} by calling `bcm_tunnel_terminator_config_add (config_key, config_action)` with the following parameters:
   - `config_key.dip6` – Configure tunnel DIP as part of the lookup key. (should the same as in `bcm_tunnel_terminator_create(unit, tunnel)`)
   - `config_key.dip6_mask` – Configure the DIP mask
   - `config_key.vrf` – Configure VRF value, this VRF is resolved after ETH. termination or previous IP/MPLS-Tunnel and is part of the three possible tunnel terminator key
   - `config_key.tunnel_id` – Global tunnel LIF ID, allocated from the `tunnel_terminator_create` API.
   - `config_action.tunnel_class` – An intermediate object to pass between the two steps of tunnel termination configuration (managed by user).

   It is possible to have multiple intermediate objects for different IPv6-P2P
2. Create an IPv6-P2P Tunnel termination object properties, see IPv4 tunnel termination configuration flow.
3. Call `bcm_tunnel_terminator_create(unit, tunnel)`
   - `tunnel.flags` – Configure P2P tunnel terminator LIF object use the `BCM_TUNNEL_TERM_USE_TUNNEL_CLASS` flag. All other flags listed in the IPv4 tunnel termination section..

- `tunnel.vrf` – Configure the VRF value. This VRF is resolved after ETH termination or previous IP/MPLS-Tunnel and is part of the three possible tunnel terminator key. combinations lookup keys.
- `tunnel.sip6` – Configure tunnel SIP as part of the lookup key.
- `tunnel.sip6_mask` – Configure tunnel SIP mask.
- `tunnel.type` – Configure the tunnel type of the tunnel to be terminated. All three IPv6 tunnel types as in MP tunnel terminator are supported
- `tunnel.tunnel_class` – An intermediate object that was created in the `bcm_tunnel_terminator_config_add` API (for the DIP).
- Additional flags and APIs for IPv6 Tunnel termination object properties, see IPv4 tunnel termination configuration flow.

30.3.2.3 IPv6-P2P Tunnel Terminator with MP Defined Object

To support tunnel P2P Tunnel terminator, together with MP defined object use the following calling sequence:

1. Create MP object for {DIP, VRF} by calling `bcm_tunnel_terminator_create(unit, tunnel)` (see parameters in Section 30.3.2.2, IPv6-P2P Tunnel Terminator Object)
2. Create IPv6-P2P Tunnel termination by calling `bcm_tunnel_terminator_create(unit, tunnel)` – See parameters in Section 30.3.2.2, IPv6-P2P Tunnel Terminator Object except
   - `tunnel.tunnel_class (Result)` – As intermediate object serves the `tunnel_id` that was created in the first call of API `bcm_tunnel_terminator_create` (for the DIP, VRF).

30.3.3 Shell Commands

None

30.3.4 Application Reference

Example of basic tunnel terminator API usage for GRE4oIPv6 tunnel:

- **Type:** CINT reference
- **Path:** `$SDK/src/examples/dnx/tunnel/cint_dnx_ipv6_tunnel_termination_basic.c`

Additional tunnel termination scenarios for additional keys and with different forwarding headers (MPLS, IPv4, IPv6):

- **Type:** CINT reference
- **Path:** `$SDK/src/examples/dnx/tunnel/cint_dnx_ipv6_tunnel_termination.c`

30.4 API Descriptions

API Name	Highlights
`bcm_tunnel_terminator_create()`	Create Ingress tunnel and set its properties.
`bcm_tunnel_terminator_delete()`	Destroy Ingress tunnel given LIF-ID.
`bcm_tunnel_terminator_get()`	Retrieve Ingress tunnel properties given Tunnel LIF-ID.
`bcm_tunnel_terminator_traverse()`	Traverse over all Ingress tunnel LIF objects.
`bcm_tunnel_terminator_config_add()`	Create P2P Ingress IPV6 tunnel.
`bcm_tunnel_terminator_config_get()`	Get P2P Ingress IPV6 tunnel.
`bcm_tunnel_terminator_config_delete()`	Destroy P2P Ingress IPV6 tunnel entry configuration.

Chapter 31: IP Tunnel v6 Encapsulation

31.1 Introduction

IPv6 encapsulation application defines the means by which packets are switched into an IPv6 domain. This section provides information on how to configure IPV6 tunnel and what actions are taken after the forwarding on the main header (MPLS/IP). API calls for defining IPV6 egress tunnel are also described.

For L2 IPv6 tunnel encapsulation cases, see Chapter 32, VXLAN IP Overlay.

It is recommended to read this section after the following sections:

- Chapter 21, Ethernet Bridge and Chapter 22, IP Router to get an overview of the basic L2 and L3 objects.
- Section 22.6, ETH-RIF (L3 VSI Interface) and VRF Definition. This section provides information about configurations that are related to next hop information, namely FEC and Egress-ARP. Also, it provides information relevant for the MyMAc operation which is a prerequisite for IP tunnel forwarding and termination.

31.2 Application Configuration Checklist

1. Set L3 port properties. For more information, see Section 22.3, Port Routing Properties.
2. Set ETH-RIF properties and create MyMac per ETH-RIF. For more information, see Chapter 22, IP Router.
3. Create Egress object: Egress-ARP For more information, see Section 22.7, L3 Egress Object: Egress-ARP (Next-Hop).
4. Egress L3 IP-tunnel IPv6 encapsulation object.

31.3 Egress L3 IP-Tunnel IPv6 Encapsulation Object

IP Tunnel encapsulation with or without additional headers is done at the Encap stages in the device. The procedure is done according to the tunnel LIF object retrieved from the EEDB. The LIF object may be pointed by the ingress databases (such as Forwarding databases, FEC/ECMP, and so on) or from egress databases (such as Egress-ACL) or from the EEDB itself by a previous MPLS/IP Tunnel. The next hop information (ARP) is usually pointed by the tunnel LIF object.

The following L3 IP-Tunnel IPv6 types are supported:

- GreAnyIn6 – IP header with GRE header. Following GRE types are supported:
  - Gre4oIPv6 header with payload MPLS/IPv4/IPv6.
  - Gre8oIPv6 with entropy key and payload MPLS/IPv4/IPv6.
- UDP – IP header with UDP header. Two different flavors are supported:
  - UDP next protocol is not VXLAN.
  - UDP next protocol is VXLAN (and VXLAN-GPE). For such a support see Chapter 32, VXLAN IP Overlay.
- IpAnyIn6 – IP header encapsulation is done without any additional headers (GRE/UDP)

Because both egress L3 IP-Tunnel IPv4 and IPv6 encapsulation objects have many attributes in common, only the differences between IPv6 and IPv4 are provided. For full information on IPv4, see Section 29.3, Egress L3 IP-Tunnel IPv4 Encapsulation Object.

NOTE: For an L3 IP-Tunnel IPv6 encapsulation object, allocate two entries in the EEDB. For instance, if the object is created with bcmEncapAccessTunnel2 encap access then Tunnel2 and Tunnel3 are occupied and cannot be used by the next pointed object (for example, MPLS-Tunnel).

31.3.1 SOC Properties

None

31.3.2 Configuration Flow

To create an egress IP tunnel, call `bcm_tunnel_initiator_create(unit, &intf, &tunnel)`

- `tunnel.dip6` – Configure the tunnel DIP to be encapsulated.
- `tunnel.sip6` – Configure the tunnel SIP to be encapsulated.
- `tunnel.encap_access` – Specify from which encapsulation stage the tunnel encapsulation procedure will start (see the table in Chapter 29, IP Tunnel v4 Encapsulation).
- `tunnel.type` – Configure the tunnel type of the tunnel to be encapsulated. Supported IPv6 tunnel types for encapsulation are:

Supported Types	Description
`bcmTunnelTypeIpAnyIn6`	Used for configuration of all IPv6 tunnels.
`bcmTunnelTypeGreAnyIn6`	Used for for configuration of all GreoIpv6 tunnels. The GRE8 tunnel encapsulation is configured with `BCM_TUNNEL_INIT_GRE_KEY_USE_LB flag`
`bcmTunnelTypeUdp6`	Used for configuration of UDPoIPv6 tunnels. Source and destination port are not required for configuration, they are resolved according to the forwarding layer type.
`bcmTunnelTypeSR6`	Used for configuration of SRv6 tunnels as support layer for SRv6 extension over IPv6

- Other object attributes are the same as with IPv4 (see Section 29.3, Egress L3 IP-Tunnel IPv4 Encapsulation Object).

31.3.3 Shell Commands

None

31.3.4 Application Reference

Example of basic-encapsulation API usage for GRE4oIPv6 tunnel:

- **Type:** CINT reference
- **Path:** `$SDK/src/examples/dnx/cint_dnx_ipv6_tunnel_encapsulation_basic.c`

Additional tunnel initiator scenarios in combination with different forwarding headers (MPLS, IPv4, IPv6):

- **Type:** CINT reference
- **Path:** `$SDK/src/examples/dnx/cint_dnx_ipv6_tunnel_encapsulation.c`

31.4 API Descriptions

API Name	Highlights
`bcm_tunnel_initiator_create()`	Create Egress IPv6 tunnel and set its properties.
`bcm_tunnel_initiator_clear()`	Destroy Egress IPv6 tunnel initiator given LIF-ID.
`bcm_tunnel_initiator_get()`	Retrieve Egress IPv6 tunnel properties given Tunnel LIF-ID.
`bcm_tunnel_initiator_traverse()`	Traverse over all IPv6 and IPv4 Egress tunnel LIF objects.

Chapter 32: VXLAN IP Overlay

32.1 Introduction

VXLAN is an L2 VPN bridging application common in data centers that have cores in an IP routing domain. In VXLAN IP overlay, VPNs are created across the data center by encapsulation of Ethernet, IP or MPLS packets with IP and UDP headers. Within the data center core, the packets are IP routed. The hosts are organized into VPN. The core data-center-core is made of an hierarchical IP networks of TOR (Top-of-Rack) routers, aggregation routers, and core routers. Packets are switched within the local VPN that contains a local TOR router. Then the packets are encapsulated with IP and UDP headers and routed across the core to the TOR router adjacent to the destination. The packet IP and UDP headers are decapsulated, and the packet is switched locally.

The BCM88690 supports two different types of VXLAN headers: VXLAN and VXLAN GPE. This chapter provides information about how to configure VXLAN (GPE) ingress and egress tunnels, what actions are taken prior to the forwarding on the main header (MPLS/IP/ETH), and how the encapsulation is treated at egress.

It is recommended to read this section after the following sections:

- Chapter 21, Ethernet Bridge and Chapter 22, IP Router to get an overview of the basic L2 and L3 objects.
- Section 22.6, ETH-RIF (L3 VSI Interface) and VRF Definition.
- Chapter 22, IP Router provides information about configurations that are related to next hop information, namely FEC and Egress-ARP. Also, it provides information relevant for the MyMAc operation which is a prerequisite for IP tunnel forwarding and termination.
- IP tunnel related sections, giving more details on the tunnel APIs and the additional possible configurations: IP Tunnel IPv4 Termination, IP Tunnel IPv6 Termination, IP Tunnel IPv4 Encapsulation, IP Tunnel IPv6 Encapsulation.

When packets enter the VXLAN VPN, they enter through TORs and are encapsulated with VXLAN overlay headers. Inside the VPN, the next hop destination is decided based on the DIP of the overlay tunnel. Upon reaching the host, packets are decapsulated by the TOR device and switched/routed locally.

32.2 Application Configuration Checklist

- Set L3 port properties – For more information, see Section 22.3, Port Routing Properties.
- Set ETH-RIF properties and create MyMac per ETH-RIF – For more information, see Section 22.6, ETH-RIF (L3 VSI Interface) and VRF Definition
- VXLAN L2-VPN.
- VXLAN Network domain.
- Ingress VXLAN-tunnel termination object.
- Egress VXLAN-tunnel encapsulation object.
- Set L3 Egress object: Ingress-FEC – For more information, see Chapter 22, IP Router.
- VXLAN Port object.
- Split Horizon filtering for VXLAN – For more information, see Section 20.2, Split Horizon (Network Groups).
- Configure L3 Forwarding entry - host/ route – For more information, see Chapter 22, IP Router.
- Configure L2 MACT UC Forwarding – For more information, see Chapter 21, Ethernet Bridge.
- Configure L2 Flooding groups – For more information, see Chapter 21, Ethernet Bridge.

32.3 VXLAN L2-VPN

This section covers the mechanism of creating a VXLAN VPN instance. VXLAN VPN provides the VSI properties required for VXLAN application which includes:

- Adding mapping between the VNI and VPN in both ingress and egress.
- Creating the VSI and configuring VSI properties such as flooding groups (for unknown UC, unknown MC and broadcast packets) for the VPN instance.

32.3.1 SOC Properties

None

32.3.2 Configuration Flow

To allocate and create a VSI, map VNI to VSI and VSI to VNI, configure MC groups for unknown destinations call the API: `bcm_vxlan_vpn_create(unit, info)`

- `info.vpn` – Virtual switching instance (VSI) ID.
- `info.vnid` – Specify the global VPN ID. This is the VNI from the packet, used to map the packet to the L2VPN.
  This parameter has a special value. The `BCM_VXLAN_VNI_INVALID` value clears the VNI-to-VSI mapping. Use this value on the update functionality with one of the following values:
  - `BCM_VXLAN_VPN_REPLACE`
  - `BCM_VXLAN_VPN_UNKNOWN_UCAST_REPLACE`
  - `BCM_VXLAN_VPN_UNKNOWN_MCAST_REPLACE`
  - `BCM_VXLAN_VPN_BCAST_REPLACE`
  - `BCM_VXLAN_VPN_WITH_ID`

  Another possible usage is to create either asymmetrical ingress or egress mapping or such with network domain !=0. In those cases, the calling sequence should be `bcm_vxlan_vpn_create` with value `BCM_VXLAN_VPN_INVALID`, followed by `bcm_vxlan_network_domain_config_add` with the desired mapping information.
- `info.match_port_class` – Configure the network domain identifier. It is used for differentiating VNI per network. Set to 0.
- `info.broadcast_group` – Configure the broadcast group
- `info.unknown_multicast_group` – Configure the unknown multicast group
- `info.unknown_unicast_group` – Configure the unknown unicast group
- `info.flags` – The flags in the following table are supported

Supported Flags	Description
`BCM_VXLAN_VPN_WITH_ID`	create VXLAN VPN with specific VPN ID
`BCM_VXLAN_VPN_WITH_VPNID`	create VXLAN VPN with specific VNI ID
`BCM_VXLAN_VPN_UNKNOWN_UCAST_REPLACE`	Replace `unknown_unicast_group` for given VXLAN VPN.
`BCM_VXLAN_VPN_UNKNOWN_MCAST_REPLACE`	Replace `unknown_multicast_group` for given VXLAN VPN.
`BCM_VXLAN_VPN_BCAST_REPLACE`	Replace `broadcast_group` for given VXLAN VPN.
`BCM_VXLAN_VPN_REPLACE`	Replace the already allocated VXLAN VPN entry with new information. This flag can be used only with `BCM_VXLAN_VPN_WITH_ID flag` and valid VNID, or `BCM_VXLAN_VPN_INVALID flag` set.

32.3.3 Shell Commands

None

32.3.4 Application Reference

Example of VXLAN VPN configuration

- **Type:** CINT reference
- **Path:** `$SDK/src/examples/sand/cint_vxlan_encapsulation.c`

32.4 VXLAN Network Domain

The BCM88690 supports two network domains for VXLAN: VSI and VRF. The VSI network domain is used when the forwarding header after VXLAN/VXLAN-GPE is Ethernet. The API can also update the VSI-to-VNI mapping, already set by `bcm_vxlan_vpn_create`. The VRF network domain is mapped to VNI when the forwarding header after VXLAN-GPE is IP.

The purpose of this API is to create a specific mapping between the VNI and VSI for an L2 network domain. To create a VSI-to-VN mapping with a network domain (identifier) equal to 0, the following two options are possible:

- Use the regular VPN creation API with `match_port_class = 0` with a valid VNID.
- Call the regular `bcm_vxlan_vpn_create` with `vnid = BCM_VXLAN_VNI_INVALID`, then configure the mapping using `bcm_vxlan_network_domain_config_add` with `network_domain = 0`. VSI and VNI are set to the desired values. Calling `bcm_vxlan_vpn_create` prior to `bcm_vxlan_network_domain_config_add` is necessary because it allocates VSI-related resources.

To create a mapping with a network domain other than 0, only one option is available.

Call `bcm_vxlan_vpn_create` with `vnid = BCM_VXLAN_VNI_INVALID`. Then configure the mapping by using `bcm_vxlan_network_domain_config_add`.

A mapping created with `bcm_vxlan_vpn_create` can be removed or retrieved using only its respective API. The same rule applies to a mapping created by `bcm_vxlan_network_domain_config_add`. This is also true for entries that are being traversed.

NOTE: The mapping is asymmetric. It can be ingress only or egress only.

32.4.1 SOC Properties

None

32.4.2 Configuration Flow

To set VNI-to-VSI or VNI-to-VRF mapping call the API `bcm_vxlan_network_domain_config_add(unit, config)`

- `config.vsi` – Virtual switch instance. Populated for L2 mapping only.
- `config.vrf` – Virtual switch instance. Populated for L3 mapping only.
- `config.network_domain` – Network identifier. Used to distinguish between multiple networks.
- `config.vni` – VNID of the VXLAN packet.
- `config.stat_id` – Egress statistics object id. Populated for counting per VNI
- `config.stat_pp_profile` – Egress statistics pp profile. Populated for counting per VNI

- `config.flags` – The following flags are supported

Supported Flags	Description
BCM_VXLAN_NETWORK_DOMAIN_CONFIG_INGRESS_ONLY	Add ingress only mapping
BCM_VXLAN_NETWORK_DOMAIN_CONFIG_EGRESS_ONLY	Add egress only mapping
BCM_VXLAN_NETWORK_DOMAIN_CONFIG_L2_MAPPING	Configure L2 mapping.
BCM_VXLAN_NETWORK_DOMAIN_CONFIG_L3_MAPPING	Configure L3 mapping.
BCM_VXLAN_NETWORK_DOMAIN_CONFIG_REPLACE	Replace the VNI-to-VSI or VNI-to-VRF mapping. It is also required on creation when the network domain is 0.

32.4.3 Shell Commands

None

32.5 Ingress VXLAN Tunnel Termination Object

VXLAN tunnel termination is done as part of the tunnel termination processing at the device. The tunnel termination procedure is done right after the first ETH header termination (if My-MAC is set). Tunnel termination requires a lookup match and a resulting InLIF for VXLAN.

The following VXLAN tunnel termination types are supported:

- Vxlan – Used for VXLAN tunnel termination. The processing is done as in IPv4 tunnels with two additional headers: UDP and VXLAN. The forwarding domain is resolved after successful tunnel termination. For ETH as the forwarding header, the forwarding domain is VSI, and the forwarding is performed in FWD blocks based on the (native) ETH header.
- VxlanGpe – Used to indicate the next header after the VXLAN. It can be: ETH, IPv4, IPv6, and MPLS. Similar to VXLAN, VXLAN-Gpe is terminated as a regular IPv4 tunnel with two additional headers: UDP and VXLAN. The forwarding domain is resolved after successful tunnel termination. For ETH as the forwarding header, the forwarding domain is VSI. For IP, the network domain is VRF. The forwarding is performed in FWD blocks based on the ETH, IPv4, IPv6, or MPLS forwarding header.
- Vxlan6 – Similar to the VXLAN type, but the main tunnel type is IPV6.Tunnel termination is performed according to IPv6 with two additional headers.
- VxlanGpe6 – Similar to VXLAN-Gpe type, but the main tunnel type is IPv6.Tunnel termination is performed according to IPv6 with two additional headers.

Since both Ingress L3 IP-Tunnel IPv4 and VXLAN termination objects have many attributes in common, this document provides only the differences of VXLAN compared to IPv4. For details about IPv4, see Section 28.3, Ingress L3 IP-Tunnel IPv4 Termination Object.

For more IP tunnel related configurations, see Section 28.3, Ingress L3 IP-Tunnel IPv4 Termination Object and Section 30.3, Ingress L3 IP-Tunnel v6 Termination Object.

32.5.1 SOC Properties

None

32.5.2 Configuration Flow

To create an ingress VXLAN tunnel terminator, call `bcm_tunnel_terminator_create(unit, tunnel)`

- `tunnel.type`: Configure the tunnel type of the tunnel to be terminated. The following table describes the supported VXLAN tunnel types for termination.

Supported Types	Description
`bcmTunnelTypeVxlan`	used for configuration of IPv4 tunnel with UDP and VXLAN as additional headers.
`bcmTunnelTypeVxlanGpe`	used for configuration of IPv4 tunnel with UDP and VXLAN-GPE as additional headers.
`bcmTunnelTypeVxlan6`	used for configuration of IPv6 tunnel with UDP and VXLAN as additional headers.
`bcmTunnelTypeVxlanGpe6`	used for configuration of IPv6 tunnel with UDP and VXLAN-GPE as additional headers.

- All other fields and additional properties APIs are the same as with an IPv4/IPv6 object.

32.5.3 Shell Commands

None

32.5.4 Application Reference

Example of Basic tunnel terminator API usage for VXLAN tunnel

- **Type:** CINT reference
- **Path:**
  - `$SDK/src/examples/sand/cint_vxlan_tunnel_termination_basic.c`
  - `$SDK/src/examples/dnx/tunnel/cint_dnx_ipv6_tunnel_termination.c`

Example of Basic tunnel terminator API usage for VXLAN-GPE tunnel

- **Type:** CINT reference
- **Path:** `$SDK/src/examples/sand/cint_ip_tunnel_termin_basic.c`

32.6 Egress VXLAN Tunnel Encapsulation Object

IPv4/IPv6 tunnel encapsulation with UDP and Vxlan/VXLAN-GPE additional headers is done at the Encap stages in the device. The procedure is done according to the tunnel LIF object retrieved from the EEDB. The LIF object may be pointed by the ingress databases (such as Forwarding databases, FEC/ECMP, and so on) or from egress databases (such as Egress-ACL) or from the EEDB itself by a previous MPLS/IP Tunnel. The next hop information (ARP) is usually pointed by the tunnel LIF object.

The following VXLAN tunnel types are supported:

- Vxlan – IPv4 header with UDP and VXLAN additional headers encapsulation. As payload only ETH is supported.
- VxlanGpe – IPv4 header with UDP and VXLAN-GPE additional headers encapsulation. The payload could be IP, MPLS, ETH.
- Vxlan6 – IPv6 header with UDP and VXLAN additional headers encapsulation. As payload only ETH is supported.
- VxlanGpe6 – IPv6 header with UDP and VXLAN-GPE additional headers encapsulation. The payload could be IP, MPLS, ETH.

Because both egress VXLAN-tunnel encapsulation objects have many attributes in common with IPv4 and IPv6 respectively, this section provides only the differences of VXLAN compared to IPv4 and IPv6. For full information on IPv4 and IPv6, see Section 29.3, Egress L3 IP-Tunnel IPv4 Encapsulation Object and Section 31.3, Egress L3 IP-Tunnel IPv6 Encapsulation Object.

32.6.1 SOC Properties

None

32.6.2 Configuration Flow

To create an egress IP tunnel, call `bcm_tunnel_initiator_create(unit, &intf, &tunnel)`

- `tunnel.type`: Configure the tunnel type of the tunnel to be encapsulated. The following table describes the supported IPv6 tunnel types for encapsulation.

Supported Types	Description
`bcmTunnelTypeVxlan`	Used for encapsulation of IPv4 tunnel with UDP and VXLAN as additional headers.
`bcmTunnelTypeVxlanGpe`	Used for encapsulation of IPv4 tunnel with UDP and VXLAN-GPE as additional headers.
`bcmTunnelTypeVxlan6`	Used for encapsulation of IPv6 tunnel with UDP and VXLAN as additional headers.
`bcmTunnelTypeVxlanGpe6`	Used for encapsulation of IPv6 tunnel with UDP and VXLAN-GPE as additional headers.

- All other fields and additional API properties are the same as IPv4/IPv6 object.

NOTE: The egress Native-PCP and DEI Remark profile comes from the tunnel.

To add VXLAN to MACT call `bcm_l2_addr_add` with `port` being the `vxlan_port_id`. See Chapter 21, Ethernet Bridge.

32.6.3 Shell Commands

None

32.6.4 Application Reference

Example of Basic tunnel encapsulation API usage for VXLAN tunnel

- **Type:** CINT reference
- **Path:** `$SDK/src/examples/sand/cint_vxlan_encapsulation.c`
  `$SDK/src/examples/sand/cint_vxlan6.c`

Example of Basic tunnel encapsulation API usage for VXLAN-GPE tunnel

- **Type:** CINT reference
- **Path:** `$SDK/src/examples/dnx/vxlan/cint_vxlan_roo_basic.c`

32.7 VXLAN Port

User should configure VXLAN port in order to update existing VXLAN tunnel lif entry with forwarding domain assignment, learning information and forwarding group. The API is used for VXLAN and VXLAN-GPE only if the forwarding is based on ETH header, for IP and MPLS forwarding, it is not needed. Learning could be configured in 3 different manners:

- FEC only
- Symmetric tunnel OutLIF + port
- FEC for ARP + Symmetric tunnel OutLIF

For VXLAN tunnel termination, if the mapping between the VNI and forwarding domain is missed, a default VPN (for VXLAN) or default VRF (for VXLAN-GPE) can be configured.

NOTE: `default_vrf` is done in the Tunnel-Termination object.

For VXLAN tunnel encapsulation, if the forwarding domain to VNI mapping is missed, the packet will be trapped with the trap-code `BCM_RX_TRAP_EG_TX_TRAP_ID_DROP`.

32.7.1 SOC Properties

None

32.7.2 Configuration Flow

To configure VXLAN Port call the API `bcm_vxlan_port_add(unit, l2vpn, &vxlan_port)`, where:

- `vxlan_port.match_tunnel_id` – gport of the tunnel termination object.
- `vxlan_port.egress_if` – FEC/ECMP pointing to ip tunnel encapsulation of type VXLAN. When the learning mode is FEC only.
- `vxlan_port.egress_tunnel_id` – gport of the tunnel initiator entry if the learning mode is symmetric OutLIF and port or symmetric OutLIF and ARP.
- `vxlan_port.criteria` – BCM_VXLAN_PORT_MATCH_VN_ID
- `vxlan_port.network_group_id` – Orientation of the vxlan port. Used for split horizon filtering.
- `vxlan_port.default_vpn` – Default VPN value. Used when there is no existing mapping between VNI and VPN.
- `vxlan_port.flags` – The flags in the following table are supported.

Supported Flags	Description
`BCM_VXLAN_PORT_WITH_ID`	create VXLAN port with specific port ID
`BCM_VXLAN_PORT_EGRESS_TUNNEL`	Indicates simple IP-OutLIF, and that the *`egress_tunnel_id`* tunnel is valid. For Primary/ Secondary pair and ECMP, the IP tunnels are defined in *`egress_if`* and *`egress_tunnel_id`* is invalid.
`BCM_VXLAN_PORT_EGRESS_OBJECT`	Indicates that *`egress_if`* is a valid object. For learning mode : tunnel OutLIF + ARP fec, *`egress_if`* is the ARP fec.
`BCM_VXLAN_PORT_REPLACE`	Replace an already allocated vxlan port entry. The flag can be used only with the *`BCM_VXLAN_VPN_WITH_ID`* flag and a valid *`vxlan_port_id`*.

32.7.3 Shell Commands

None

32.7.4 Application Reference

Example of VXLAN port configuration and API usage

- **Type:** CINT reference
- **Path:** `$SDK/src/examples/sand/cint_vxlan_general.c`

32.8 UDP Destination Port

Users can update the UDP destination port. The UDP destination port indicates the next UDP protocol. For VXLAN tunnel termination, this feature updates the expected UDP destination port value to identify VXLAN header. For VXLAN tunnel encapsulation, this feature indicates the UDP destination port value to use.

If not configured, the following default UDP destination port applies:

- 4789 for VXLAN
- 4790 for VXLAN-GPE

32.8.1 SOC Properties

None

32.8.2 Configuration Flow

To configure the UDP destination port, call `bcm_switch_control_set(unit, type, arg)`. The parameters are as follows:

- `type` – `bcmSwitchVxlanUdpDestPortSet` for VLXAN or `bcmSwitchVxlanGpeUdpDestPortSet` for VXLAN-GPE
- `arg` – The new value for the UDP destination port

32.8.3 Shell Commands

None

32.8.4 Application Reference

None

32.9 Native Egress VLAN Editing

Egress VLAN editing on the native Ethernet header is supported for VXLAN and VXLAN ROO applications. The native egress VLAN editing information is configured as part of the VSI-to-VNI mapping and has its own virtual object in the SDK. The `bcm_vlan_port_create` virtual AC handles the VLAN editing information in the VSI-to-VNI mapping.

VPN object creation and VSI-to-VNI mapping are handled by `bcm_vxlan_vpn_create` and `bcm_vxlan_network_domain_add`.

Depending on which API is called first, the VSI-to-VNI mapping can be created by the `bcm_vxlan_vpn_create`, `bcm_vxlan_network_domain_add`, or `bcm_vlan_port_create` API.

32.9.1 SOC Properties

None

32.9.2 Configuration Flow

The `bcm_vxlan_network_domain` API creates the VSI-to-VNI mapping.

1. Create VXLAN VPN instance with `vnid = BCM_VXLAN_VNI_INVALID`.
   See Section 32.3, VXLAN L2-VPN.
2. Create a new VSI-to-VNI mapping with `bcm_vxlan_network_domain_add`.
   See Section 32.4, VXLAN Network Domain.
3. Create an AC with `bcm_vlan_port_create`.
4. Configure VLAN editing with `bcm_vlan_port_translation_set`.
5. Delete the AC with `bcm_vlan_port_destroy`.
6. Delete the VSI-to-VNI mapping with `bcm_vxlan_network_domain_remove`.

The `bcm_vxlan_vpn_creates` API creates the VSI-to-VNI mapping.

1. Create VXLAN VPN instance with vnid = `BCM_VXLAN_VNI_INVALID`.
   See Section 32.3, VXLAN L2-VPN.
2. Create VSI-to-VNI mapping and the VNI-to-VSI mapping with `bcm_vxlan_vpn_create` with `flags=BCM_VXLAN_VPN_REPLACE` and a valid VNID.
3. Create an AC with `bcm_vlan_port_create`.
4. Configure VLAN editing with `bcm_vlan_port_translation_set`.
5. Delete both VSI-to-VNI and VNI-to-VSI mapping with `bcm_vxlan_vpn_create` with `flags = BCM_VXLAN_VPN_REPLACE` and `vnid = BCM_VXLAN_VNI_INVALID`.

The `bcm_vlan_port_create` API creates the VSI-to-VNI mapping and VNI is configured with the `bcm_vxlan_network_domain` API.

1. Create a VXLAN VPN instance with `vnid = BCM_VXLAN_VNI_INVALID`.
   See Section 32.3, VXLAN L2-VPN.
2. Create an AC with `bcm_vlan_port_create`. Creates the VSI-to-VNI mapping.

3. Configure VLAN editing.
4. Update the VSI-to-VNI mapping with `bcm_vxlan_network_domain_add` with the `BCM_VXLAN_NETWORK_DOMAIN_CONFIG_REPLACE` flag.
   See Section 32.4, VXLAN Network Domain.
5. Delete the AC and the VSI mapping by using `bcm_vlan_port_destroy`.

NOTE: It is not necessary to call `bcm_vxlan_network_domain_remove` here.

The `bcm_vlan_port_create` API creates the VSI-to-VNI mapping, and VNI is configured with the `bcm_vxlan_vpn_create` API.

1. Create a VXLAN VPN instance with `vnid = BCM_VXLAN_VNI_INVALID`.
   See Section 32.3, VXLAN L2-VPN.
2. Create an AC.
3. Use `bcm_vlan_port_create` to create the VSI-to-VNI mapping.
4. Configure VLAN editing with `bcm_vlan_port_translation_set`.
5. Update VSI-to-VNI mapping and create the VNI-to-VSI mapping by using `bcm_vxlan_vpn_create` with `flags=BCM_VXLAN_VPN_REPLACE` and a valid VNID.
6. Delete VNI-to-VSI mapping by using `bcm_vxlan_vpn_create` with `flags=BCM_VXLAN_VPN_REPLACE` and `vnid = BCM_VXLAN_VNI_INVALID`.
7. Delete the AC and the VSI mapping with `bcm_vlan_port_destroy`.

32.9.3 Shell Commands

None

32.9.4 Application Reference

Example of VXLAN with native VLAN egress editing configuration:

- **Type**: CINT reference
- **Path**: `$SDK/src/examples/dnx/vxlan/cint_vxlan_roo_basic.c`

32.10 Geneve Overlay

The Generic Network Virtualization Encapsulation (Geneve) is very similar to VXLAN because both of these features are network encapsulation protocols.

32.10.1 Configuration Flow

The configuration flow is similar to VXLAN. The only difference is the tunnel type for tunnel termination and encapsulation.

32.10.2 Ingress Geneve Tunnel Termination

See Ingress VXLAN Tunnel Termination.

The following table lists the supported tunnel type.

Supported Types	Description
`bcmTunnelTypeGeneve`	Used for configuration of IPv4 tunnel with UDP and Geneve as additional headers.

32.10.3 Egress Geneve Tunnel Encapsulation

See Egress VXLAN Tunnel Encapsulation.

The following table lists the supported tunnel type.

Supported Types	Description
`bcmTunnelTypeGeneve`	Used for configuration of IPv4 tunnel with UDP and Geneve as additional headers.

32.10.4 Application Reference

Example of Geneve

- Type: CINT reference
- Path: `$SDK/src/examples/dnx/cint_dnx_geneve_tunnel.c`

32.11 API Descriptions

32.11.1 VXLAN VPN

API Name	Highlights
`bcm_vxlan_vpn_destroy()`	Destroy and deallocate the given VSI, unmap VNI to VSI.
`bcm_vxlan_vpn_create()`	Allocate and create a VSI, map VNI to VS**I.**
`bcm_vxlan_vpn_get()`	Get VXLAN VPN information.

32.11.2 Ingress VXLAN Tunnel Termination Object

API Name	Highlights
`bcm_tunnel_terminator_create()`	Create Ingress tunnel and set its properties.
`bcm_tunnel_terminator_clear()`	Destroy Ingress tunnel given LIF-ID.
`bcm_tunnel_terminator_get()`	Retrieve Ingress tunnel properties given Tunnel LIF-ID.
`bcm_tunnel_terminator_traverse()`	Traverse over all Ingress tunnel LIF objects.

32.11.3 Egress VXLAN Tunnel Encapsulation Object

API Name	Highlights
`bcm_tunnel_initiator_create()`	Create egress VXLAN tunnel and set its properties.
`bcm_tunnel_initiator_delete()`	Destroy egress VXLAN tunnel initiator given LIF-ID.
`bcm_tunnel_initiator_get()`	Retrieve egress VXLAN tunnel properties given Tunnel LIF-ID.
`bcm_tunnel_initiator_traverse()`	Traverse over all VXLAN Egress tunnel LIF objects.

32.11.4 VXLAN Port

API Name	Highlights
`bcm_vxlan_port_delete()`	Destroy already created VXLAN port.
`bcm_vxlan_port_add()`	Create VXLAN port.
`bcm_vxlan_port_get()`	Get VXLAN port information.
`bcm_multicast_vxlan_encap_get()`	Get the encapsulation information from the VXLAN port.

32.11.5 VXLAN Network Domain

API Name	Highlights
`bcm_vxlan_network_domain_config_add()`	Add VNI to VSI or VNI to VRF ingress or egress only mapping.
`bcm_vxlan_network_domain_config_get()`	Get local identifier (vsi or vrf) with VNI mapping.
`bcm_vxlan_network_domain_config_remove()`	Remove local identifier (vsi or vrf) with VNI mapping.
`bcm_vxlan_network_domain_config_traverse()`	Traverse local identifier (vsi or vrf) with VNI mapping.

32.11.6 UDP Destination Port

API Name	Highlights
`bcm_switch_control_set()`	Update UDP destination port for VXLAN or VXLAN-GPE.

Chapter 33: Operations, Administration, and Maintenance

33.1 Introduction

The DNX family devices perform Operations, Administration, and Maintenance (OAM) functions through the following entities:

- The OAM classifier within the packet processing-pipe: used for trapping OAM traffic according to LIF and MD-Level (Ethernet OAM).
- The OAM Processor (OAMP) block within the device: may be used to manage OAM traffic.
- The generic counter processors and time of day (ToD) functions of the processing pipe.

This chapter starts with an application checklist and is then followed by a detailed description of each application and its associated APIs.

33.2 Application Configuration Checklist

- Traps
- OAM profiles and classification
- My-CFM-MAC configuration
- MPLS/PWE channel type configuration
- OAMP MEP-DB management
- Up MEP configuration
- LBM response configuration
- OAMP punt packets
- OAM events and protection packets
- Creating OAM groups (MA)
- Creating endpoints and remote endpoints
- Loss/delay measurement support
  - Hierarchical LM
  - Performance monitoring through OAMP
  - SLM through OAMP
- OAM primary VLAN

33.3 Traps

The OAM classifier eventually assigns traps to OAM packets according to the packet's MD-Level and LIF and the endpoints configured on the LIF. All OAM traps must thus be configured before traffic arrives. The following traps must be configured:

The following traps are required for the full OAM application:

- `bcmRxTrapOamEthAccelerated` – Ethernet OAM accelerated endpoints
- `bcmRxTrapOamY1731MplsTp` – OAM over MPLS-TP accelerated endpoints
- `bcmRxTrapOamY1731Pwe` – OAM over PWE, Section MPLS accelerated endpoints
- `bcmRxTrapBfdOamDownMEP` – default trapping for packets not processed in OAMP
- `bcmRxTrapOamLevel` – MD-Level filter (packet's MD-Level is below that of a configured MEP on a LIF)
- `bcmRxTrapOamPassive` – Passive side filter
- `bcmRxTrapSnoopOamPacket` – Down MEP snoop (MIP CCM, LTM packets)
- `bcmRxTrapOamPerformanceEthAccelerated` – LM/DM trapping to the OAMP for Ethernet OAM accelerated endpoints.
- `bcmRxTrapOamPerformanceY1731MplsTp` – LM/DM trapping to the OAMP for MPLS-TP OAM accelerated endpoints.
- `bcmRxTrapOamPerformanceY1731Pwe` – LM/DM trapping to the OAMP for PWE OAM accelerated endpoints.
- If additional destinations are required, user-defined traps may be used.

For the full calling sequence, see Chapter 12, Traps. `bcm_rx_trap_set()` must be called with the following parameters:

- `trap_config.flags` must be set with the following flags:
  - `BCM_RX_TRAP_TRAP` – Set the system headers of the trapped packet as expected by the OAMP.
  - `BCM_RX_UPDATE_DEST` – Update the trapped packet's destination.
  - `BCM_RX_TRAP_UPDATE_FORWARDING_HEADER` – Update the forward-header-offset, required for packets processed at the OAMP
- Assuming the OAMP is used for processing CCMs, the traps `bcmRxTrapOamEthAccelerated`, `bcmRxTrapOamY1731MplsTp` and `bcmRxTrapOamY1731Pwe`, `trap_config.dest_port` must be set to the local OAMP. For the remaining traps, the destination should be the CPU or any other OAM processing unit.
- `egress_forwarding_index` should be set to 7.

The CPU can inject packets into the ingress using the `tx` shell command. If the packet is supposed to be trapped to the OAMP and processed in it, configure the trap differently to allow overwriting the parsing index with value 1. When using the default traps, this can be done by calling the function `field_oam_layer_index_main()` in the CINT file `cint_field_oam_layer_index.c`.

33.3.1 SOC Properties

None

33.3.2 Configuration Flow

For each trap, `bcm_rx_trap_type_create()` should be called, followed by `bcm_rx_trap_set()`. Traps must then be used in the `destination` field in the `bcm_oam_result_t` struct in `bcm_oam_profile_action_set()` API.

33.3.3 Shell Commands

None

33.3.4 Application Reference

All traps are configured under `appl_dnx_oam_trap_init()` under `appl_dnx_oam_init()`. If destinations other than the CPU/OAMP are needed, `appl_dnx_trap_init()` may be updated.

The reconfiguration of the traps to support a packet injected from the CPU to the ingress is in the function `field_oam_layer_index_main()` in the CINT file `cint_field_oam_layer_index.c`. It can also be used as an example to reconfigure user-defined traps used for this purpose.

33.4 OAM Profiles and Classification

On MIPs and MEPs, OAM classification and trapping is done per profile. Four APIs are required to create one endpoint.

- `bcm_oam_profile_create` – Create a profile and determine the LM and DM type on the profile. There are four types of profiles: ingress LIF, egress LIF, ingress accelerated, egress accelerated.
- `bcm_oam_profile_action_set` – This determines the actions (i.e. traps, counter-enable/disable, and so on) per profile.
- `bcm_oam_lif_profile_set` – This sets an egress/ingress LIF profiles on LIFs. Error level and passive side trapping will be defined according to the egress/ingress LIF profile.

  Those OAM profiles can also be unset with a valid gport and both profiles are set to `BCM_OAM_PROFILE_INVALID`. If the LIF is used by MEPs, delete the MEPs before unsetting the profiles.
- `bcm_oam_endpoint_create` – Finally, endpoints are set on a given `gport` representing a LIF.
  - For non accelerated endpoints, the trapping behavior will be taken according to the egress or ingress LIF profile defined on the `gport`.
  - For accelerated endpoints, the `bcm_oam_endpoint_create()` API is given an accelerated profile parameter, `acc_profile_id`.

In other words actions are decoupled from endpoints. A more detailed calling sequence, including illustrations follows.

33.4.1 SOC Properties

None

33.4.2 Configuration Flow

Configuration is done through the following steps.

1. Create OAM profiles in the ingress (`bcmOAMProfileIngressLIF`) and egress (`bcmOAMProfileEgressLIF`) with `bcm_oam_profile_create()`.
   a. In this API, LM type must be determined. This is done through the flags `BCM_PROFILE_LM_TYPE_DUAL_ENDED` or `BCM_OAM_PROFILE_LM_TYPE_NONE`. Default is single ended LM.
   b. DM type must also be determined. Default is 1588, to use NTP, set the flag `BCM_OAM_PROFILE_DM_TYPE_NTP`. `BCM_OAM_PROFILE_DM_TYPE_NONE` is also available if no stamping is required.
   c. In addition, for accelerated MEPs, create an `bcmOAMProfileIngressAcceleratedEndpoint` or `bcmOAMProfileEgressAcceleratedEndpoint` profile, depending on the direction of the endpoint.

      Accelerated MEPs need 3 profiles – ingress or egress accelerated profiles for trapping packets that match the MEP's LIF/MD-Level/Direction and ingress, egress profiles for MD-Level error packets and passive side trapping.
2. Set up the profile behavior according to each different `oam_action_key` using `bcm_oam_profile_action_set()`. It is possible to disable stamping by using the `BCM_OAM_ACTION_NONE` flag.

3. Associate the profile to a `gport` with `bcm_oam_lif_profile_set()`
   a. Both ingress and egress profiles are required, even if only one endpoint will be set on that LIF[1]. For example if a Down MEP is set on a LIF then the passive side trapping will be done according to the profile at the egress and the active according to the ingress profile
   b. For accelerated endpoints, LIF profiles must also be configured for level and passive side traps.
4. Finally, create an endpoint with `bcm_oam_endpoint_create()` on the LIF.
   a. For accelerated endpoints, supply the accelerated profile through `acc_profile_id` parameter.

In total the API `bcm_oam_profile_create()` must be called once per profile, `bcm_oam_lif_profile_set()` must be called once on each LIF, `bcm_oam_profile_action_set()` must be called per each profile and key configuration.

33.4.2.1 Non-Accelerated Endpoint Configuration and Packet Flow

Figure 17 illustrates the configuration flow for non-accelerated endpoints, per direction.

- `bcm_oam_lif_profile_set()` is responsible for mapping a LIF to a lif-profile.
- `bcm_oam_profile_action_set()` is responsible for mapping the following:
  - `profile` – Ingress or egress LIF profile
  - `endpoint_type` – May be `bcmOAMMatchTypeMIP`, `bcmOAMMatchTypeMEP`, `bcmOAMMatchTypePassive`, `bcmOAMMatchTypeLowMDL`, or `bcmOAMMatchTypeLowMDLMIP`. The key type `bcmOAMMatchTypeLowMDL` affects packets below MIPs and MEPs. To update only packets below MIP, first call `bcm_oam_profile_action_set()` with the key `bcmOAMMatchTypeLowMDL`, then call it with the key `bcmOAMMatchTypeLowMDLMIP`.
  - `OpCode` – OAM PDU's OpCode. All opcodes are mapped into one of 16 available OpCode profiles. For information about the default OpCode profile mapping and OpCode profile management, see Section 33.6, OpCode Mapping.
  - `Dest_MAC_type` – May be `bcmOAMDestMacTypeMcast`, `bcmOAMDestMacTypeUknownUcast`, or `bcmOAMDestMacTypeMyCfmMac`. See My-CFM-MAC Configuration chapter for information about resolution.
  - inject - Used to distinguish between injected packets and packets trapped on the passive side to
    - `destination` – A trap gport with snoop strength and forwarding strength.
    - `counter_increment` – Determines whether the given OAM PDU will increment LM counters.
- Calls to `bcm_oam_endpoint_create()` then determines the actual endpoints configured on the LIF. Per packet's MD-Level, this determines the resolution of `endpoint_type` when a packet is trapped.

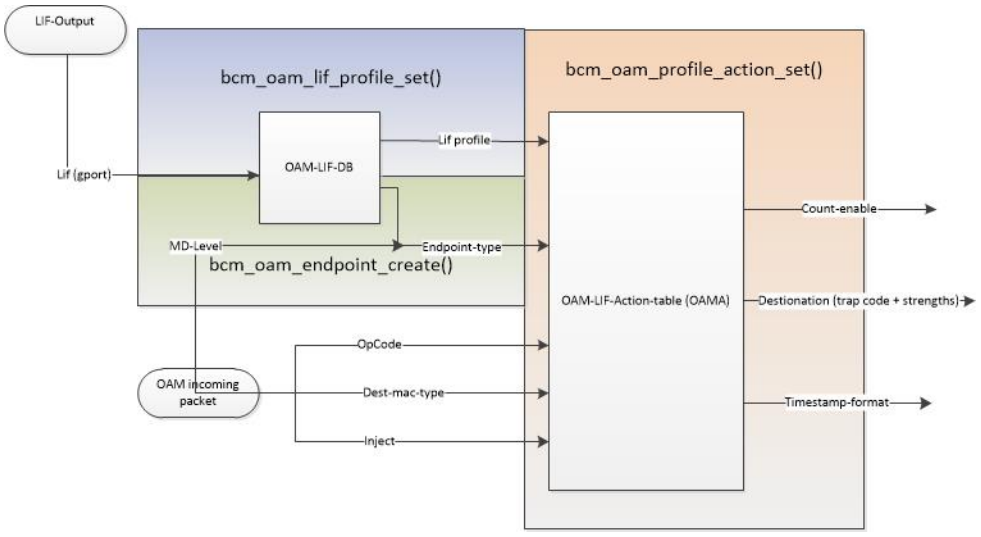
When an OAM PDU arrives, the following mappings are then made:

1. Previous blocks produces a LIF. Ingress/egress OAM-LIF-DB is accessed with that LIF.
2. According to calls to `bcm_oam_lif_profile_set()`, the OAM-LIF-DB produces a LIF-profile.
3. According to calls to `bcm_oam_endpoint_create()` on the LIF**,** along with the direction (ingress or egress) and the packet's MD-Level, the packet's endpoint-type is determined. For example if Down MEPs were created on a LIF with MD-Levels 6, 4 and a packet arrived with MD-Level 5 at the ingress, the endpoint-type will be `bcmOAMMatchTypeLowMDL`**.** If on the same configuration a packet arrived on the egress on the same LIF with MD-Level 6 the endpoint-type will be `bcmOAMMatchTypePassive`**.**
4. According to the packet's DA and calls to `bcm_l2_station_add()` (see Section 33.5, My-CFM-MAC Configuration) to determine dest-mac-type.
5. OpCode is determined according to the packet's OpCode.
6. Inject is set if the packet was injected.

1. The sole exception for this is MPLS/PWE endpoints. Endpoints on PWE/MPLS may omit the egress or the ingress part of the classifier.

7. Finally, a packet may be assigned a `trap_code` along with trap-strength and snoop strength as well as counter-increment command according to calls to `bcm_oam_profile_action_set()` with the key fields filled with values assigned to the packet in Step 2 through Step 6.

Figure 17: Non Accelerated Endpoint Configuration and Packet Flow



33.4.2.2 Accelerated Endpoint Configuration and Packet Flow

Figure 18 illustrates the additional configurations required for accelerated endpoints:

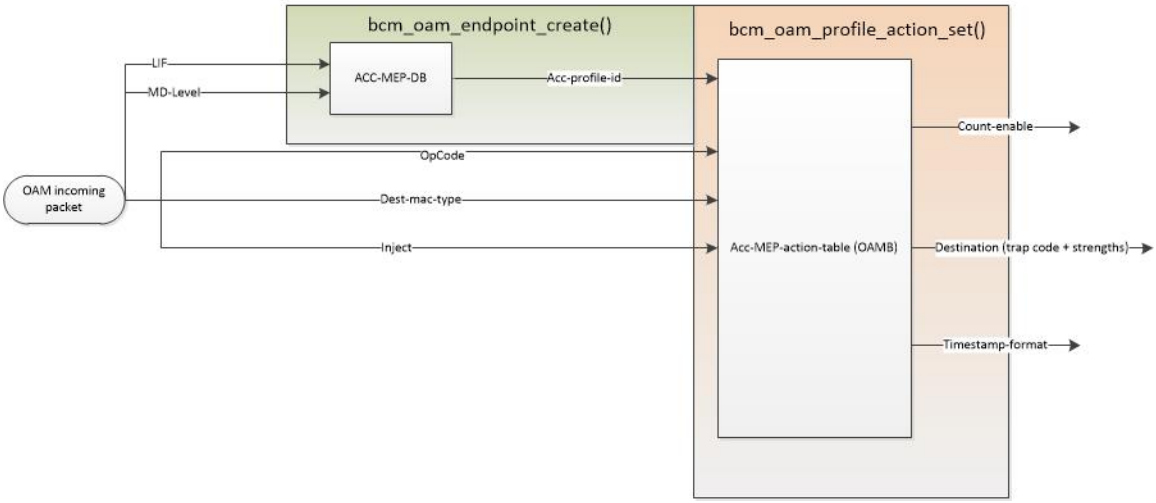
- `bcm_oam_profile_action_set()` is identical to the non-accelerated case except for the following exceptions:
  - `profile` – Ingress accelerated or egress accelerated profile.
  - `endpoint_type` must be `bcmOAMMatchTypeMEP`.
  - `counter_increment` may also be used on accelerated profiles, but it should be configured on the LIF profile. The counter increment indication on both profiles acts as a bitwise AND.
- Calls to `bcm_oam_endpoint_create()` then determines the actual endpoints configured on the LIF and maps it to an `acc_profile_id`.

When an OAM PDU arrives, the following mappings are then made:

1. Previous blocks produces a LIF. Along with the packet's MD-Level and direction (ingress/egress), ingress/egress ACC-MEP-DB is accessed.
2. If an accelerated endpoint was created on that LIF/MD-Level, direction combination with `bcm_oam_endpoint_create()`, an Acc-profile-ID is produced.

   Otherwise, the packet follows the non-accelerated-flow.
3. OpCode, Dest-Mac-type, inject are determined as in the non-accelerated case.
4. Trap code, trap/snoop strengths, counter increment commands are assigned to the packet as in step 7 in the non accelerated case.

Figure 18: Accelerated Endpoint Configuration and Packet Flow



33.4.2.3 Inject Entries

Inject entries affect packets injected by the OAMP, CPU, and FPGA. These should be set to control counter-increment commands per packets. For example, for dual-ended LM (CCM based LM), CCMs should not increment counters, while for single-ended LM, CCMs should be counted. Besides setting counter-increment on trapped packets, it is necessary to set counter-increment accordingly on injected packets for TX counters.

Inject entries affect Up MEP packets and *Down MEP egress injected* packets, not including standard down MEP entries. The entries should reside on the opposite side of the MEP. For example, for an Up MEP on LIF X, inject entries should be set at the ingress on LIF X. Note that the same profile will also be used for Down MEP trapping on the same LIF.

On a profile set on the MEP's LIF (for an Up MEP it would be an ingress-LIF profile, for a Down MEP an egress-LIF profile), call `bcm_oam_profile_action_set()` with the following parameters:

- In the `bcm_oam_action_key_t struct` set:
  - `opcode`: OpCode that should be updated
  - `endpoint_type`: should `bcmOAMMatchTypePassive` and `bcmOAMMatchTypeMEP`
  - `dest_mac_type` should be `bcmOAMDestMacTypeMcast`, and possibly `bcmOAMDestMacTypeMyCfmMac`, `bcmOAMDestMacTypeUknownUcast`
  - inject should be 1
- In the `bcm_oam_action_result_t` set:
  - Destination to `BCM_GPORT_INVALID`
  - `counter_increment` to the desired value.

`bcm_oam_profile_action_set()` should be called once with each key combination

In addition, if an accelerated MEP exists in the opposite direction, on the same LIF, `bcm_oam_profile_action_set()` should be called in the same manner for an egress/ingress-accelerated-endpoint profile, in the opposite direction of the MEP, as before.

For OAM endpoints over PWE, MPLS-TP, and Section MPLS, the `counter_increment` behavior of packets can be controlled. See Section 33.23.4, Hierarchical LM by LIF for details.

33.4.2.4 Deleting Profiles

Profiles may be deleted by calling `bcm_oam_profile_delete()`, which clears all actions defined on a profile. Calling this API for a profile used by an existing MEP should be avoided.

33.4.2.5 Default Profile

The following table describes the behavior of the default profiles for MEPs

Packet Type	Profile Type	Trap	Destination
Down MEP CCM multicast	`bcmOAMProfileIngressAcceleratedEndpoint`	bcmRxTrapOamEthAccelerated	OAMP
Up MEP CCM multicast	`bcmOAMProfileEgressAcceleratedEndpoint`	bcmRxTrapEgTxOamUpMEPOamp	OAMP
Ingress MIP CCM multicast	`bcmOAMProfileIngressLIF`	bcmRxTrapSnoopOamPacket	Forward + Snoop to CPU
Egress MIP CCM multicast	`bcmOAMProfileEgressLIF`	None (use mirror profile). Snooped copy will come system headers	Forward + Snoop to CPU
Down MEP LMM/DMM/ AIS/LTM/APS/SLM multicast/unicast	`bcmOAMProfileIngressAcceleratedEndpoint` and `bcmOAMProfileIngressLIF`	bcmRxTrapBfdOamDownMEP	CPU
Up MEP LMM/DMM/AIS/ LTM/APS/SLM multicast/ unicast	`bcmOAMProfileEgressAcceleratedEndpoint` and `bcmOAMProfileEgressLIF`	bcmRxTrapEgTxOamUpMEPDest1 (Trap code on packet at the CPU will be 0xe0)	CPU
Down MEP LBM unicast	`bcmOAMProfileIngressAcceleratedEndpoint` and `bcmOAMProfileIngressLIF`	bcmRxTrapOamReflector	Reflector (LBR will be sent)
Up MEP LBM unicast	`bcmOAMProfileEgressAcceleratedEndpoint` and `bcmOAMProfileEgressLIF`	bcmRxTrapEgTxOamReflector	Reflector (LBR will be sent)
Down MEP LBM multicast	`bcmOAMProfileIngressAcceleratedEndpoint` and `bcmOAMProfileIngressLIF`	bcmRxTrapBfdOamDownMEP	CPU
Up MEP LBM multicast	`bcmOAMProfileEgressAcceleratedEndpoint` and `bcmOAMProfileEgressLIF`	bcmRxTrapEgTxOamUpMEPDest1 (Trap code on packet at the CPU will be same as bcmRxTrapOamEthAccelerated)	CPU
CCM unicast Down MEP/ MIP	`bcmOAMProfileIngressAcceleratedEndpoint` and `bcmOAMProfileIngressLIF`	bcmRxTrapOamLevel	CPU
CCM unicast Up MEP/MIP	`bcmOAMProfileEgressAcceleratedEndpoint` and `bcmOAMProfileEgressLIF`	bcmRxTrapEgTxOamLevel (Trap code on packet at the CPU will be same as bcmRxTrapOamLevel)	CPU
Other, unicast Down MEP	`bcmOAMProfileIngressAcceleratedEndpoint` and `bcmOAMProfileIngressLIF`	bcmRxTrapBfdOamDownMEP	CPU
Other unicast Up MEP	`bcmOAMProfileEgressAcceleratedEndpoint` and `bcmOAMProfileEgressLIF`	bcmRxTrapEgTxOamUpMEPDest1 (Trap code on packet at the CPU will be 0xe0)	CPU
LTM Down MIP	`bcmOAMProfileIngressLIF`	bcmRxTrapBfdOamDownMEP	CPU
LTM Up MIP	`bcmOAMProfileEgressLIF`	bcmRxTrapEgTxOamUpMEPDest1 (Trap code on packet at the CPU will be 0xe0)	CPU
Other, unicast Down MEP/ MIP, incorrect level, and unkown unicast	`bcmOAMProfileIngressAcceleratedEndpoint` and `bcmOAMProfileIngressLIF`	bcmRxTrapOamLevel	CPU

Packet Type	Profile Type	Trap	Destination
Other unicast Up MEP/MIP, incorrect level, and unkown unicast	`bcmOAMProfileEgressAcceleratedEndpointbcmOAMProfileEgressLIF`	bcmRxTrapEgTxOamLevel (Trap code on packet at the CPU will be same as bcmRxTrapOamLevel)	CPU
Passive match	`bcmOAMProfileIngressAcceleratedEndpointbcmOAMProfileIngressLIF`	bcmRxTrapOamPassive	—
Passive match	`bcmOAMProfileEgressAcceleratedEndpointbcmOAMProfileEgressLIF`	bcmRxTrapEgTxOamPassive (Trap code on packet at the CPU will be same as bcmRxTrapOamPassive)	—

More profiles and examples are defined in cint_oam_action.c:

- `oam_set_opcode_destination_to_oamp()` – For a given OpCode and profile, set destination to the OAMP.
- `dnx_oam_action_set()` – Create trap according to destination (may be local/system port) and call `bcm_oam_profile_action_set()`
- `oam_set_of_action_profiles_create()` – Create a list of profiles of different LM types, for Ethernet CCM, MPLS and PWE endpoints.

33.4.3 Shell Commands

- `OAM ClassiFier` displays the mappings in the classifier per output or per last packet.
- `OAM Last_lookUp` displays the last lookup in the ingress/egress ACC-MEP-DBs and OAM-LIF-DBs.

33.4.4 Application Reference

The OAM application in `appl_dnx_oam_configure_default_profile()` and `appl_dnx_oam_configure_default_acc_profile()` configures default profiles, one for each type (ingess X LIF, egress X LIF, and accelerate). An example of how these may be used can be found in `oam_example()` in `cint_dnx_oam.c`.

33.5 My-CFM-MAC Configuration

When entering the OAM classifier, every packet is assigned a Dest-MAC-type. There are three options:

1. Multicast – DAs starting with 01.
2. My-CFM-MAC – Packet's DA matches My-CFM-MAC that is assigned to a port.
3. Unknown unicast – Other types of packets.

Assigning My-CFM-MAC on a port can be done through `bcm_l2_station_add()`.

33.5.1 SOC Properties

None

33.5.2 Configuration Flow

Call `bcm_l2_station_add` with the following parameters:

- `flags`: Set to `BCM_L2_STATION_OAM`
- `dst_mac`: Set to the required DA.
- `dst_mac_mask`: must be: FF:FF:FF:FF:FF:FF
- `src_port`: Set to the required port
- `src_port_mask`: Set to -1
- All other fields are not required

`station_id` is an output parameter that may be used for get/delete API.

The following limitations apply:

- The API sets the My-CFM-MAC on a port on both directions
- Per port, the five most significant bytes of the DA must be constant
- The least significant byte is defined by a range, as follows:
  - When adding My-CFM-MAC on a port the first time, both the minimum LSB and maximum LSB are the LSB of the added My-CFM-MAC. When adding L2 addresses, the LSB range is extended.
  - Deleting My-CFM-MAC from a port involves the following process:
    - Let LSB_D be the LSB of the deleted My-CFM-MAC address.
    - The minimum LSB is set to LSB_D + 1 if LSB_D is closer to the minimum LSB than the maximum LSB.
    - The maximum LSB is set to LSB_D – 1 if LSB_D is closer to the maximum LSB than the minimum LSB.
    - The entire range is cleaned if LSB_D is equal to the maximum LSB.
    - This information applies to the BCM88830, BCM88480 and BCM88800 only.

An option for adding and deleting the My-CFM-MAC LSB range directly by calling `bcm_l2_station_add` with the following parameters:

- `flags`: Set to BCM_L2_STATION_OAM | BCM_L2_STATION_OAM_RANGE
- `dst_mac`: Set to the required DA
- `dst_mac_mask`: Must be FF:FF:FF:FF:FF:FF
- `src_port`: Set to the required port
- `src_port_mask`: Set to -1
- `my_cfm_mac_lsb_range_min`: Set to 0~255
- `my_cfm_mac_lsb_range_max`: Set to 0~255
- All other fields are not required.
- `station_id` - Output parameter that may be used for get/delete API.

The following limitations apply:

- The API sets the My-CFM-MAC on a port on both directions
- Per port, the five most significant bytes of the MSB DA must be constant
- The least significant byte is defined by a range [my_cfm_mac_lsb_range_min, my_cfm_mac_lsb_range_max]:
  - This applies to the BCM88800, BCM88480, and BCM88830 only.
- `bcm_l2_station_delete()` configuration for BCM88800, BCM88480, and BCM88830:
  - If the input LSB of the MAC address in `bcm_l2_station_delete()` is closer to the minimum configured LSB, the new range will be configured-max to (input LSB + 1).
  - If the input LSB is equal to the maximum of the existing range, the range is cleared.
  - If the input LSB of the MAC address in `bcm_l2_station_delete()` is closer to the maximum configured LSB, the new range will be (input LSB - 1) to configured-max.

33.5.3 Shell Commands

None

33.5.4 Application Reference

In `oam_example()` in `cint_dnx_oam.c` My-CFM-MACs are set and may be used as an example.

33.6 OpCode Mapping

Any OpCode from an OAM packet is mapped into an OpCode profile, and the classification is performed according to the OpCode profile.

However, `bcm_oam_profile_action_set` configures the action for a specific OAM OpCode by retrieving its OpCode profile. Therefore, OAM OpCode mapping of the OAM OpCode should be configured prior to any `profile_action_set` that uses the given OpCode.

The following table shows the default OAM OpCode mapping.

Table 93: Default OAM OpCode Mapping

OAM PDU	OAM PDU OpCode	OpCode Profile
BFD or Data	—	0
CCM	1	1
LBR	2	2
LBM	3	3
LTR	4	4
LTM	5	5
LMR	42	6
LMM	43	7
DMR	46	8
DMM	47	9
1DM	45	10
SLM	55	11
SLR	54	12
AIS	33	13
LINEAR_APS	39	14
OTHER	Other	15

33.6.1 SOC Properties

None

33.6.2 Configuration Flow

To configure the OpCode profile map, call `bcm_oam_opcode_map_set()` with the following parameters:

- `opcode`: OAM PDU OpCode
- `profile`: Opcode profile (0-15).

To get the current OpCode mapping, call `bcm_oam_opcode_map_get()` with the following parameters:

- `opcode`: OAM PDU OpCode
- `*profile`: Pointer to OpCode profile to get the currently assigned profile..

33.6.3 Shell Commands

None

33.6.4 Application Reference

None

33.7 MPLS/PWE Channel Type Configuration

Dedicated APIs are used to associate different Channel-Types from the PW Associated Channel with OAM or BFD over PWE/MPLS-TP. The user must configure these before sending/receiving traffic. For example the user may configure Channel-Type 0x8902 with BHH OAM and 0x22 with BFD.

33.7.1 SOC Properties

None

33.7.2 Configuration Flow

The API `bcm_oam_mpls_tp_channel_type_rx_set()` must be used to configure Channel Types used for trapping and `bcm_oam_mpls_tp_channel_type_tx_set()` must be used for Channel Types used for OAMP transmission.

When attempting to destroy a channel type through `bcm_oam_mpls_tp_channel_type_tx_delete()`, the default values are restored as follows:

- 0x8902 for `bcmOamMplsTpChannelY1731`
- 0x27 for `bcmOamMplsTpChannelPweonoam`
- 0x22 for `bcmOamMplsTpChannelMplsTpBfd`
- 0x7 for `bcmOamMplsTpChannelPweBfd`.

To delete the values altogether, the defaults must then be deleted.

33.7.3 Shell Commands

None

33.7.4 Application Reference

Initial Channel Types are configured in the OAM reference application code through `dnx_tune_oam_mpls_tp_channel_type_rx_init()` and `dnx_tune_oam_mpls_tp_channel_type_tx_init()` in `oam_tune.c`.

33.8 RX Up MEP Configuration

The sequence in the configuration flow describes how to configure Up MEP traps, including PMF programs that are required.

33.8.1 SOC Properties

None

33.8.2 Configuration Flow

33.8.2.1 Egress Trapping

Trap allocation is done through the following steps:

1. Create an ingress user-defined trap with `bcm_rx_trap_type_create()` and `bcm_rx_trap_set()`. Destination of the trap should be OAMP, CPU, and so on.
2. Set up an egress trap, again using `bcm_rx_trap_type_create()` and `bcm_rx_trap_set()`. In the former API, the trap type should be one of the following:
   - `bcmRxTrapEgTxOamLevel`
   - `bcmRxTrapEgTxOamPassive`
   - `bcmRxTrapEgTxOamUpMEPDest1` – For trapping the CPU.
   - `bcmRxTrapEgTxOamUpMEPDest2` – For trapping to all other destinations that are not OAMP or the CPU. For example, Accelerator (for LBRs, TST), OAMP server, and so on. Currently, only one of these applications may be used at a time.
   - `bcmRxTrapEgTxOamUpMEPOamp`

   In the latter API, in the trap config struct, the following should be set:
   - `stamped_trap_code` – Trap code that get stamps on the packet's system headers. For packets destined to the OAMP, use the trap code associated with `bcmRxTrapOamEthAccelerated` (0xe0). This trap code may be used for all trapped packets for Jer1 compatibility.
   - `cpu_trap_gport` – Ingress trap from step 1, including strength, using the `BCM_GPORT_TRAP_SET()` macro. Ingress trap strength must be greater than 7 to bypass various ingress filters.

   The egress trap ID is the `trap_id` output from `bcm_rx_trap_type_create()`.
3. Set up a PMF rule which does not add additional system headers. The ETPP will build system headers with an FHEI and the trap code from `stamped_trap_code`. This is only mandatory for packets destined for the OAMP. If the CPU/ customer equipment knows how to process packets with two sets of system headers then this stage can be skipped.

   For example if the trap in Step 1 is created with the flag `BCM_RX_TRAP_TRAP`, then the packet will be trapped to the destination defined in `bcm_rx_trap_set()` with two sets of system headers:
   - The outer set will have a sniff FHEI with the trap code from Step 1.
   - The inner set will also have a sniff FHEI, with the trap code being `stamped_trap_code` from Step 2, and the trap-qualifier being the OAM-ID.
4. Finally, the egress trap id should be used in the destination field in `bcm_oam_profile_action_set()` (in the `bcm_oam_action_result_t`) struct. The destination should be used as a GPORT encoding, along with the trap strength, using the macro `BCM_GPORT_TRAP_SET()`.

The preceding steps should be used in some variations for each of the ETPP OAM traps:

- `bcmRxTrapEgTxOamUpMEPOamp`: In this case, perform Step 1 to Step 4. In Step 1, the destination of the trap should be the OAMP.
- `bcmRxTrapEgTxOamUpMEPDest1`: In this case, perfrom Step 1, Step 2, and Step 4. Step 3 is optional. The destination of the trap in step 1 should be the CPU or external OAM processing unit. If an additional destination is required then this should be repeated for `bcmRxTrapEgTxOamUpMEPDest2`.
- `bcmRxTrapEgTxOamLevel, bcmRxTrapEgTxOamPassive`. In these cases, only Step 2 and Step 4 are recommended. The `cpu_trap_gport` from Step 2 should be the trap associated with `bcmRxTrapOamLevel` and `bcmRxTrapOamPassive` respectively. In this case, Step 3 *must not* be performed.

33.8.2.2 Egress Snooping

The following sequence may be used to configure egress snooping:

1. Create a snoop command using the API `bcm_mirror_destination_create()` API. Set the following fields in `bcm_mirror_destination_t`:
   – `flags` set to `BCM_MIRROR_DEST_IS_SNOOP | BCM_MIRROR_DEST_EGRESS_ADD_ORIG_SYSTEM_HEADER`
   – `gport` – Snoop destination
   To ensure there is an FHEI (see Step 4), set `packet_control_updates.valid` to `BCM_MIRROR_PKT_HEADER_UPDATE_FABRIC_HEADER_EDITING`
2. Extract the snoop command from the above API through the field `mirror_dest_id` using the macro `BCM_GPORT_MIRROR_GET()`. For example:
   `egr_snoop_trap_id = BCM_GPORT_MIRROR_GET(mirror_dest.mirror_dest_id);`
3. The snoop command should be used in the destination field in `bcm_oam_profile_action_set()` (in the `bcm_oam_profile_action_result_t`) struct. The destination should be used as a GPORT encoding, along with the snoop strength, using the macro `BCM_GPORT_TRAP_SET()`.
4. Additionally, the API `bcm_mirror_header_info_set()` should be called to ensure that packets are trapped to the CPU with a trap FHEI. For more information of the system header stack trapped to the CPU, see Section 33.16.3, Hierarchical Loss Measurement.

Up MEP is not supported with the SOC property `pph_learn_extension_disable=0` on trapped Up MEP packets.

Up MEP is not supported with valid DSP extensions on trapped Up MEP packets.

33.8.3 Shell Commands

None

33.8.4 Application Reference

As an example, the application initializes the UP-MEP traps, see `appl_dnx_oam_trap_init()` and `appl_dnx_oam_up_mep_trap_init()`.

For an example of a PMF rule for system headers, see `appl_dnx_field_trap_no_system_header()` function.

33.9 OAM Ethernet Loopback

OAM Ethernet Loopback (LB) is a protocol out of a set of protocols defined in IEEE 802.1ag Ethernet CFM (Connectivity Fault Management). LB allows to detect fault location, bandwidth reliability and jitter by sending LB messages to a MEP or MIP.

Packet with unicast LBM frame sent to local MEP or MIP (with DA equal to MY-CFM-MAC) will be transmitted back to the sender with swapped SA and DA and with LBR OpCode instead of LBM.

There are two types of reflectors: Up MEP and Down MEP.

Up MEP

1. The packet gets classified in the egress and marked as reflector packet.
2. SA and DA are swapped, and OpCode is updated.
3. The packet gets recycled with a PTCH.
4. The packet should then be forwarded according to the VLAN + in-port (from PTCH, extracted from the packet's original destination).

Down MEP

1. The packet gets classified in the ingress and assigned a trap code.
2. Ingress PMF redirects packet to its origin.
3. Trap code updates the OutLIF for loopback.
4. Through the OutLIF, the egress pipeline swaps the SA with the DA and updates the OpCode.

33.9.1 Application Configuration Checklist

33.9.1.1 Unicast Down MEP

1. OAM loopback OutLIF – Allocate OutLIF for OAM loopback.
2. OAM down MEP loopback trap – Allocate trap code for OAM down MEP loopback and set trap to override OutLIF to the allocated OAM loopback OutLIF.
3. OAM Ingress Classifier – Configure OAM ingress classifier to override trap-code for unicast OAM with MY-CFM-MAC as destination and OpCode LBM
4. Field processor (ACL) group – Create a field processor group that identifies the OAM down MEP loopback packets (according to the fixed trap code `bcmRxTrapOamReflector` set in OAM ingress classifier) and overrides the destination to be source-system-port.

33.9.1.2 Unicast Up MEP

1. OAM up MEP loopback trap – Allocate egress trap code for OAM up MEP loopback and optionally configure whether low/high priority recycle context should be used.
2. OAM Egress Classifier – Configure OAM egress classifier to override trap-code for unicast OAM with MY-CFM-MAC as destination and OpCode LBM.

33.9.2 SOC Properties

None

33.9.3 Configuration Flow

1. OAM loopback OutLIF – To allocate OutLIF for unicast reflector call the API:

   `bcm_oam_reflector_encap_create(unit, flags &encap_id)`

   – `flags` – set to `BCM_OAM_REFELCTOR_ENCAP_WITH_ID` to allocate a specific OutLIF or 0 otherwise.
   – `encap_id` – (IN) Specify the required OutLIF in a case flag `BCM_OAM_REFELCTOR_ENCAP_WITH_ID` set. (OUT) the allocated OutLIF.

2. OAM down MEP loopback trap:

   a. Allocate custom trap code by calling to API:

      `bcm_rx_trap_type_create(unit, flags, type, &trap_id)`

      - `flags` – set to 0
      - `type` – set to `bcmRxTrapOamReflector`
      - `trap_id` – will be set to the allocated trap ID.

   b. Set trap to override the OutLIF to be the OAM loopback OutLIF by calling to API:

      `bcm_rx_trap_set(unit, trap_id, &config)`

      - `trap_id` – set to trap ID previously allocated
      - `config.encap_id` – set to the reflector OutLIF gport. Converting from OutLIF interface (which is the output of the API `bcm_oam_reflector_encap_create`) to OutLIF gport done by macro `BCM_L3_ITF_LIF_TO_GPORT_TUNNEL`
      - `config.flags` – Set to `BCM_RX_TRAP_UPDATE_ENCAP_ID`

   See Chapter 12, Traps, for general details about traps.

3. OAM up MEP loopback trap:

   a. Allocate a custom trap code by calling to API:

      `bcm_rx_trap_type_create(unit, flags, type, &trap_id)`

      - `flags` – set to 0
      - `type` – set to `bcmRxTrapEgTxOamReflector`
      - `trap_id` – will be set to the allocated trap ID.

   b. Set the trap by calling the API:

      `bcm_rx_trap_set(unit, trap_id, &config)`

      - `trap_id` – set to trap ID previously allocated
      - `config.is_recycle_high_priority` – set to 0 for using low priority recycle context. Refer to section "Snif" for general details about recycle context.

   See Chapter 12, Traps, for general details about traps.

4. Field Processor group for unicast down MEP loopback: Refer to Chapter 11, Field Processor for general details about field processor. Refer to the application reference for more information.

33.9.4 Shell Commands

None

33.9.5 Application Reference

Default application with detailed description of the application

- **Type:** Default application
- **Path:** `$SDK/src/appl/reference/dnx/appl_ref_oam_init.c`

NOTE:

- Unicast down MEP and up MEP loopbacks configured in default application.
- Unicast down MEP loopback configure in function `appl_dnx_oam_downmep_reflector_init()` excluding ingress OAM classifier which configured in function `appl_dnx_oam_default_profile_set()`
- Unicast up MEP loopback configure in function `appl_dnx_oam_upmep_reflector_init()` excluding egress OAM classifier which configured in function `appl_dnx_oam_default_profile_set()`

33.9.6 API Descriptions

API Name	Highlights
`bcm_oam_reflector_encap_create()`	Allocate and set OAM loopback OutLIF
`bcm_oam_reflector_encap_destroy()`	Destroy allocated OAM loopback OutLIF

33.10 OAM TST LBM Transmission

The collectors transmit LBMs and TSTs, as well as collecting statistics on incoming TSTs and LBRs.

TST and Loopback APIs have been provided. The implementation of these APIs is based almost entirely on the Accelerator APIs.

33.10.1 Configuration Flow

1. Create an MEP or use an existing MEP. The MEP must be accelerated.
2. Create TST, Accelerator, GTF, and CTF with `bcm_oam_loopback_add()`:
   - Set ID to the endpoint ID created in the first step.
   - The API returns `gtf_id` created by Accelerator to send LBM packets. If `BCM_OAM_LOOPBACK_WITH_GTF_ID` is set in flags, Accelerator will be created with a specific GTF ID. A similar process occurs with `ctf_id` and `BCM_OAM_LOOPBACK_WITH_CTF_ID`.
   - Set the peer MAC address in field `peer_da_mac_address` to be set in the loopback packet.
   - `num_tlvs` – Specify the 0, 1, and 2 TVLs in the TST packet.
   - The TLVs define the TLVs to add to the loopback packet.The type of TLVs can be any of the following:
     - `bcmOamTlvTypeNone`
     - `bcmOamTlvTypeData`
     - `bcmOamTlvTypeTestPrbsWithCRC`
     - `bcmOamTlvTypeTestPrbsWithoutCRC`
     - `bcmOamTlvTypeTestNullWithCRC`
     - `bcmOamTlvTypeTestNullWithoutCRC`.
3. Set the LBM packet bandwidth.
   - Set the field `gtf_id` created by `bcm_oam_loopback_add()`.
   - Set the rate to configure the maximum rate for TST traffic.
   - Set `max_burst` to configure the burst for TST traffic.
4. Set up traps.

   For Up MEP, the trap code `bcmRxTrapEgTxOamUpMEPDest2` must be used to trap an LBR packet to an Accelerator port. For Down MEPs, the trap code `bcmRxTrapUserDefine` traps an LBR packet to an Accelerator port. Refer to the CINT file `cint_dnx_oam_tst_lb.c`.Statistics may be retrieved with `bcm_oam_loopback_get()`.
   - Field ID configures the endpoint ID created in the previous step.
   - The API returns the transmit packet counter store in `tx_count` and stores the received packet counter in the `rx_count` field
   - Calculate the drop counter by `tx_count` – `rx_count` and store in the `drop_count` field.
   - Return the TLVs configured in the GTF flow by using the `bcm_oam_loopback_add()` API.
   - Set the rate to configure the maximum rate for TST traffic.

33.10.1.1 TST RX/TX Session

TST packet transmission and reception is performed with two sets of APIs: `bcm_oam_tst_rx_add()` and `bcm_oam_tst_tx_add()`. The configuration is analogous to the loopback API, except CTF is configured on `bcm_oam_tst_rx_add()` and GTF on `bcm_oam_tst_tx_add()`.

33.10.2 Shell Commands

None

33.10.3 Application Reference

For a CINT example, see `cint_dnx_oam_tst_lb.c`.

33.11 OAMP Punt Packet

Packets arriving at the OAMP undergo validity checks to make sure they are in the correct format. Failing packets may be punted to a CPU or any other destination for further processing.

Packets that trigger a State change have an option to be punted to the CPU as well. To prevent CPU packet flooding, it is possible to define a punting rate (sampling ratio) per RMEP. The sampling ratio applies only to packets triggering event changes (RDI or TLV).

Punt packets arrive at the CPU port (or any other destination) with a configurable trap code indicating the punt reason.

33.11.1 SOC Properties

None

33.11.2 Configuration Flow

The following punt traps are supported:

- `bcmRxTrapOampTrapErr` – Packet arrived at the OAMP with illegal trap code
- `bcmRxTrapOampTypeErr` – Packet arrived at the OAMP with illegal MEP type
- `bcmRxTrapOampRmepErr` – Packet arrived at the OAMP with RMEP index error (CCM PDU's Name does not match one of the MEPs RMEP's Name)
- `bcmRxTrapOampMaidErr` – Packet arrived at the OAMP with incorrect MAID (excluding 48B MAID endpoints)
- `bcmRxTrapOampFlexCrcMissErr` – 48B MAID Packet arrived at the OAMP with incorrect MAID (including flexible 20B and 40B MAID endpoints)
- `bcmRxTrapOampMdlErr` – Packet arrived at the OAMP with incorrect MDL. Note that such punt packet should never be observed. Packets with incorrect MDL should not be trapped to the OAMP to begin with.
- `bcmRxTrapOampCcmIntrvErr` – Packet arrived at the OAMP with incorrect CCM interval.
- `bcmRxTrapOampRmepStateChange` – OAM packet arrived that does not match the current RMEP state (Port or Interface TLV status). Applied only on remote endpoints where `sampling_ratio` is nonzero.
- `bcmRxTrapOampParityErr` – Parity error occurred in the OAMP.
- `bcmRxTrapOampTimestampErr` – Packet arrived to the OAMP with timestamp miss.

If packets come with multiple errors, the priority is as follows:

1. bcmRxTrapOampParityErr
2. bcmRxTrapOampTrapErr
3. bcmRxTrapOampTypeErr
4. bcmRxTrapOampMaidErr

5. bcmRxTrapOampMdlErr
6. bcmRxTrapOampCcmIntrvErr
7. bcmRxTrapOampRmepErr
8. bcmRxTrapOampFlexCrcMissErr

Punt packets may be configured with the standard trap APIs:

1. Call `bcm_rx_trap_type_create`.
   The `WITH_ID` flag may not be set since the trap ID must be a valid user-defined trap that is not used for anything else.
2. Allocate a `bcm_rx_trap_config_t` structure and set the `dest_port` accordingly. `dest_port` can be a gport of one of the following type:
   - `BCM_GPORT_TYPE_LOCAL`
   - `BCM_GPORT_TYPE_MODPORT`
   - `BCM_GPORT_TYPE_TRUNK`
   - `BCM_GPORT_TYPE_SYSTEM_PORT`
   - `BCM_GPORT_TYPE_UCAST_QUEUE_GROUP`
   - `BCM_GPORT_TYPE_MCAST_QUEUE_GROUP`
   - `BCM_GPORT_TYPE_MCAST`
   - `BCM_GPORT_TYPE_BLACK_HOLE`
   - `BCM_GPORT_TYPE_LOCAL_CPU`
3. Call `bcm_rx_trap_set` with the above `trap_id` and `bcm_rx_trap_config_t` structure. In the structure, the flag `BCM_RX_TRAP_TRAP` must be set. On `bcm_rx_trap_get`, the `dest_port` field will contain the gport of the trap's destination port.
4. The priority of punt packets (through the transmitted ITMH) may be configured globally through the
5. `bcm_oam_control_set()` API:
   - Set the type to be `bcmOamControlOampPuntPacketIntPri`
   - Use `value` to determine inner priority of OAMP-punted packets. Bits [1:0] of given value determine the color (DP), bits [4:2] determine the Traffic Class.

33.11.3 Shell Commands

None

33.11.4 Application Reference

All punt configurations are initialized in `appl_dnx_oam_punt_config` with the destination being the CPU.

33.12 OAMP Sampling of Good Packets

OAM packets received from a remote endpoint are monitored by the OAMP. When receiving a packet that is corrupted, the OAMP can send a punt packet to the CPU. For the purpose of monitoring, this feature provides a way to punt good packets (that is, valid packets) to the CPU.

For information about packet formats, see Section 34.7, OAMP Punt Packet.

33.12.1 SOC Properties

None

33.12.2 Configuration Flow

Configure the following punt trap:

1. Call `bcm_rx_trap_type_create`. The `WITH_ID` flag may not be set since the trap ID must be a valid user-defined trap that is not used for anything else.
2. Allocate a `bcm_rx_trap_config_t` structure and set the `dest_port` accordingly. The `dest_port` parameter can be a gport of one of the following types:
   - `BCM_GPORT_TYPE_LOCAL`
   - `BCM_GPORT_TYPE_MODPORT`
   - `BCM_GPORT_TYPE_TRUNK`
   - `BCM_GPORT_TYPE_SYSTEM_PORT`
   - `BCM_GPORT_TYPE_UCAST_QUEUE_GROUP`
   - `BCM_GPORT_TYPE_MCAST_QUEUE_GROUP`
   - `BCM_GPORT_TYPE_MCAST`
   - `BCM_GPORT_TYPE_BLACK_HOLE`
   - `BCM_GPORT_TYPE_LOCAL_CPU`
3. Call `bcm_rx_trap_set` with the above `trap_id` and `bcm_rx_trap_config_t` structure.

33.12.2.1 OAM Configuration

1. Call `bcm_oam_endpoint_create()` to create a local endpoint.
2. Call `bcm_oam_endpoint_create()` with the `BCM_OAM_ENDPOINT_REMOTE` flag set to create remote endpoint associated with the local endpoint. Mandatory parameters are as follows:

- `punt_good_packet_period` – OAM good packets sampling period in milliseconds. Every `punt_good_packet_period` milliseconds, a single packet is punted to the CPU.
- Set the `BCM_OAM_ENDPOINT_FLAGS2_PUNT_NEXT_GOOD_PACKET` flag to punt the next received good packet to the CPU. Otherwise, it is set when the period becomes 0.

33.12.2.2 BFD Configuration

Call `bcm_bfd_endpoint_create()` to create BFD endpoint. Mandatory parameters are:

- `punt_good_packet_period` – BFD good packets sampling period in milliseconds. Every punt_good_packet_period milliseconds, a single packet is punted to the CPU.
- Set `BCM_BFD_ENDPOINT_PUNT_NEXT_GOOD_PACKET` flag in order to punt the next received good packet to the CPU. Otherwise, it is set when the period becomes 0.

33.12.3 Shell Commands

None

33.12.4 Application Reference

Punt next good packet configuration is initialized by `appl_dnx_oam_punt_config` with the destination being the CPU.

33.13 OAM Events and Protection Packets

When an event is detected, an event notification is pushed into the event FIFO.

When the event FIFO is non-empty an interrupt is triggered and events are returned through callbacks.

In addition to event interrupts, an event can be packed into a protection packet. To reduce the traffic overhead, the OAMP waits 160 clock cycles after event FIFO is turned to non-empty and sends all events generated during this time in a single packet. Protection packets will be encapsulated with system headers. The size of the protection packet header depends on the number of events and can vary from 64 bytes to 192 bytes. The OAM/BFD events will be on the bottommost part of the packet. Note that all types of events will be sent to the configured destination.

The event format is shown in the following table.

Field	Bits	Definition
RMEP DB address	15:0	Pointer to RMEP DB
LoC	17:16	0 – No LoC event 1 – LoC event 2 – Almost LoC event 3 – LoC Clear event
RDI set	18	1 – RDI set event 0 – No RDI set event
RDI clear	19	1 – RDI clear event 0 – No RDI clear event
RMEP state change	20	1 – RMEP state change event 0 – No RMEP state change event
Type	22:21	0 – RMEP event 1 – N/A 2 – MEP-Rx-Report 3 – N/A
RMEP state	40:23	18-bit field that stores the RMEP new state Valid only when RMEP state change bit is set ■ Ethernet: – Remote-state[4:3] – expected remote port status – Remote-state[2:0] – expected remote interface status ■ BFD: – Remote-state[17:10] – expected detect multiplier – Remote-state[9:6] – expected flags profile – Remote-state[5:4] – expected BFD state – Remote-state[3:0] – expected diag profile
Reserved	47:41	0

In protection packets, each event as above is buffered by 2 bytes of zeros. For example, the format would be a 6-byte event, 2-byte buffer, 6-byte event, and so on.

The following limitations exist on the BCM88690, BCM88800, and BCM88480:

- OAMP can generate protection packets, which carry events generated by either the RMEP scanner (such as LOC, LOC-clear, and so on) or the RX processor (for example, RDI set/clear). These events can be in J2 format, which is 6B, but they are padded to 8B in the actual packet. In some cases, half of the 8B event will appear at the end of one protection packet and the second half will appear at the start of the next protection packet.

- When a protection packet is encountered that has a trailing 4B *half* event, the next protection packet will start with the other half of the event. The protection packet parser must be able to handle such cases so that events are not lost.

33.13.1 SOC Properties

None

33.13.2 Configuration Flow

To register a callback triggered define the callback structure:

```
typedef int (*bcm_oam_event_cb)(
    int unit,
    uint32 flags,
    bcm_oam_event_type_t event_type,
    bcm_oam_group_t group,
    bcm_oam_endpoint_t endpoint,
    void *user_data);
```

The `event_type` is one of the following:

- `bcmOAMEventEndpointCCMTimeout` – CCM timeout
- `bcmOAMEventCCMTimeoutEarly` – Early Time Out
- `bcmOAMEventEndpointCCMTimein` – CCM resumed
- `bcmOAMEventEndpointRemote` – RDI received and set
- `bcmOAMEventEndpointRemoteUp` – RDI reset
- `bcmOAMEventEndpointPortDown` – Port state down
- `bcmOAMEventEndpointPortUp` – Port state up
- `bcmOAMEventEndpointStateChange` – State change of Remote-MEP
- `bcmOAMEventEndpointInterfaceDown` – Interface Up
- `bcmOAMEventEndpointInterfaceUp` – Interface Down
- `bcmOAMEventEndpointInterfaceTesting`
- `bcmOAMEventEndpointInterfaceUnknown`
- `bcmOAMEventEndpointInterfaceDormant`
- `bcmOAMEventEndpointInterfaceNotPresent`
- `bcmOAMEventEndpointInterfaceLLDown`

Whenever an Interface state changes or a Port state changes per remote MEP, the device will generate two events: one relating to the Port state (`bcmOAMEventEndpointPortDown` or `bcmOAMEventEndpointPortUp`), and one relating to the Interface state (`bcmOAMEventEndpointInterface*`).

For events to be triggered, events must be registered using the API `bcm_oam_event_register`:

- `event_types` may be set using the macro BCM_BFD_EVENT_TYPE_SET, thus allowing for one callback to be configured for different events.

Events may be unregistered using the API `bcm_oam_event_unregister`.

Protection packets may be configured through the `bcm_rx_trap_set()` and `bcm_rx_trap_type_create()`. Sequence is identical to punt packet configuration, with the trap type `bcmRxTrapOampProtection`.

NOTE: When protection packets are not enabled, the OAMP injects invalid protection packets that are dropped in the ingress. As a result *diag counters* and various other diagnostics may report errors, however, this has no other effect.

33.13.3 Shell Commands

None

33.13.4 Application Reference

Refer to `cint_dnx_oam.c` for an example.

33.14 Creating OAM Groups (MAs)

Each OAM endpoint must be associated with a group. The groups serve two purposes:

- Groups cluster one or more OAM endpoints together into a set.
- For accelerated endpoints, groups define the MEG-ID on CCM packets. This applies to the MEG-ID on transmitted CCM packets, and MEG-ID which the OAMP verifies upon reception. (For groups composed of non accelerated endpoints, the name may be 0).

The device supports 3 MEG-ID formats.MEG-ID Format = 32 (ICC based format, defined by ITU-T Y.1731), MEG-ID Format = 3 (2 octet integer, defined by IEEE 802.1ag) or fully flexible 48B MAID:

33.14.1 ICC-Based Format

The following table shows the structure of the ICC-based format defined by ITU-T Y.1731.

Byte	Value	Interpretation
0	1	Maintenance Domain Name Format
1	32	Format= ICC-based format
2	13	Length= 13 bytes
8:3	—	6 Bytes ICC (ITU carrier code)
15:9	Value per MEP	7 Bytes UMC code
47:16	0	Unused zero padding

When an ICC-based MAID is required, create a group with the `name` field set to {1, 32,13,<6 byte ICC>, <7 byte UMC>,<zero padding>}.

33.14.2 Two-Octet Integer Format

The following table shows the structure of the two-octet integer format defined by IEEE 802.1ag.

Byte	Value	Interpretation
0	1	Maintenance Domain Name Format
1	3	Short MA Name Format
2	2	Short MA Name length
4:3	MAID, value per MEP	—
48:5	0	Unused zero padding

When a two-octet integer format is required, create a group with the *name* field set to {1, 3,2,<2 byte MAID>,<zero padding>}.

33.14.3 Fully Flexible 48-Byte MAID

Fully flexible 48B MAID is available by utilizing an additional 3 MEP-DB entries (for self-contained MEPs).

- OAMP may transmit one MEG-ID on CCMs and verify a different MEG-ID for incoming CCMs. Additional MEG-ID may be provided through `rx_name` parameter.

  It is possible to update the group name on the fly.

33.14.4 Flexible 20-Byte MAID

Flexible 20B MAID is mostly the same as fully flexible 48B MAID. The differences with 20B MAID are as follows:

- Supported on self-contained MEPs only.
- Enabled by the `BCM_OAM_GROUP_FLEXIBLE_MAID_20_BYTE` flag.
- Only the first 20-byte characters can be set. Other characters are padded with zero.

33.14.5 Flexible 40-Byte MAID

Flexible 40B MAID is mostly the same with fully flexible 48B MAID. The differences with 40B MAID are as follows:

- Supported on self-contained MEPs only.
- Enabled by the `BCM_OAM_GROUP_FLEXIBLE_MAID_40_BYTE` flag.
- Only the first 40-byte characters can be set. Other characters are padded with zero.

33.14.6 SOC Properties

None

33.14.7 Configuration Flow

Group may be created through `bcm_oam_group_create()` API. Parameters are:

- `id` – Group ID. Later used in `bcm_oam_endpoint_create()`
- `name` – MEG-ID to be used on CCM packets. Must be one of the formats above. When using non accelerated endpoints, name may be {0}
- `flags` – May be one of the following:
  - `BCM_OAM_GROUP_FLEXIBLE_MAID_48_BYTE`
  - `BCM_OAM_GROUP_FLEXIBLE_MAID_20_BYTE`
  - `BCM_OAM_GROUP_FLEXIBLE_MAID_40_BYTE`
  - `BCM_OAM_GROUP_WITH_ID.`

  Additionally, `name` or `rx_name` may be updated with `BCM_OAM_GROUP_REPLACE` for fully flexible 48B MAID groups.
- `rx_name` (optional) – May be used when incoming MEG-ID differs from the outgoing MEG-ID. If set, incoming CCM's MEG-ID are compared against this value at the OAMP. Only available when using BCM_OAM_GROUP_FLEXIBLE_MAID_48_BYTE.

33.15 Creating OAM Endpoints

When all the items described previously have been covered and defined, OAM endpoints may be created. Two types of endpoints exist:

- Non-accelerated endpoints: In this case, OAM PDUs are simply trapped according to the profiles defined on the endpoint's LIF and the endpoint's MD-Level.

- Accelerated endpoints: In this case CCMs are transmitted periodically from the OAMP and incoming CCMs may be monitored for loss of continuity (LoC). In addition, LM and DM may be managed through the OAMP, although these require further API calls. The rest of this rubric describes the various configuration options for accelerated endpoints

33.15.1 Local and Remote Endpoints

Local endpoints are used to transmit CCMs and trap multicast OAM PDUs. Multicast CCMs should be trapped to the OAMP, although this is controlled by users through the endpoint's accelerated profile. Per local endpoint, multiple remote endpoints may be defined. Remote endpoints perform the following functionalities:

- Monitoring LoC status by maintaining a timer from last received CCM. When an LoC occurs or is cleared, an event may be generated.
- Monitoring Port/Interface state of the last received CCM. When either one of the Port or Interface is updated, an event may be generated.
- Remote endpoint may also automatically control the RDI transmitted by the local endpoint.

When a CCM is trapped to the OAMP, it will be associated with a local endpoint. Within the OAMP, a mapping is done according to the local endpoint ID and the MEP-ID taken from the CCM PDU to a remote endpoint.

When a remote endpoint is created, it is initially in the *init* state and will not produce LoC events. After it receives the first CCM, it goes into time-in state. Only after that stage can an LoC event occur, although it is possible to create the endpoint directly in the time-in state.

33.15.2 Endpoint Types and Encapsulation

Endpoints may roughly be divided into two types: Ethernet OAM and MPLS/PWE. For Ethernet OAM, the complete PDU is built by the OAMP. For MPLS/PWE endpoints, only the forwarding header is built by the OAMP. The encapsulation per type is as such:

- Ethernet endpoints (`bcmOAMEndpointTypeEthernet`)
  - DA – 01:80:c2:00:00:3<Endpoint's MD-Level>
    - Unicast DA is also available (see `dst_mac_address`)
  - SA – Configured through the `src_mac_address` field.
  - VLAN tags: 0, 1 or 2 VLAN tags may be used. if no tags are required, `outer_tpid, inner_tpid` should remain blank. If only one is required then fill only `outer_tpid`. The fields `vlan, pkt_pri` determine the VLAN fields, `inner_vlan, inner_pkt_pri` determine the fields for inner VLAN tag (if necessary).
- MPLS/PWE endpoints. In such endpoints, The L2 encapsulation is built in the egress stage. The user must provide an ARP pointer through the `intf_id` field. This can be taken for example from `bcm_l3_egress_create()` API. Alternatively, a FEC may be provided. In this case, set `tx_gport` to –1 (invalid).
  - PWE endpoints (`bcmOAMEndpointTypeBHHPwe`) – In this case, the OAMP builds an ACH as well as a PWE label. the control word on the ACH may be set through `bcm_oam_mpls_tp_channel_type_tx_set()`. The PWE label is built through the `egress_label` struct, which is nested in the `endpoint_info` struct. Use the `label, ttl, exp` fields.
  - MPLS endpoints (`bcmOAMEndpointTypeBHHMPLS`) – The OAMP builds a GAL in addition to the encapsulation in `bcmOAMEndpointTypeBHHPwe` endpoints. The GAL comes between the label and the ACH, and its TTL will always be 64 and EXP 0.
  - MPLS Section endpoints (`bcmOAMEndpointTypeBhhSection)` – OAMP will build only a GAL and a ACH, as described above with the exception that TTL and EXP fields on the GAL are controlled by the `ttl, exp` fields in the `egress_label` struct.
  - PWE GAL endpoints (`bcmOAMEndpointTypeBHHPweGAL)` – In this case the encapsulation in the OAMP will be identical to `bcmOAMEndpointTypeBHHMPLS` endpoints.

- For Ethernet Down MEP and MPLS/PWE, the OAMP also builds an ITMH. The ITMH is controlled through the following fields:
  - `tx_gport` – May be a system port, MOD port, trunk port, and so on.
  - int_pri – Determines the TC/DP. Bits [1:0] determine the DP, bits [4:2] determine the TC.
- It is possible to create Ethernet Down MEPs per trunk member port instead of per LIF. This can be done as follows:
  - `gport` – Must be a local port encoded with a trunk member port.
  - `tx_gport` – Must be a system port encoded with a trunk member port.
  - To identify the OAM packet per trunk member port, a special VLAN port must be created on the trunk with flag `BCM_VLAN_PORT_FLAGS2_OAM_TRUNK_MEMBER`.

33.15.3 Port/Interface TLV Status

The OAMP may inject CCMs with Port or Interface TLVs and may monitor incoming Port and Interface TLVs per RMEP. These are both controlled through the fields `interface_state` and `port_state`. for remote endpoints, these should be read after calling `bcm_oam_endpoint_get()`. For local endpoints these should be set in `bcm_oam_endpoint_create()`.

33.15.4 Remote Endpoint Event Handling and Automatic RDI Updates

Upon LoC or state (Port/Interface TLV or RDI) event, it is possible to determine how the OAMP generates an event per RMEP as well as how this affects the RMEP's local MEP.

In the RMEP, this is done through the following `flags2` and `flags`:

1. Upon LoC, it is possible to have the local MEP transmit CCMs with the RDI bit on. This can be achieved with the flags2 `BCM_OAM_ENDPOINT_FLAGS2_RDI_ON_LOC`. By default, this feature is disabled.
2. Upon LoC clear (time-in event), it is possible to have the local MEP transmit CCMs with the RDI bit off. This can be achieved with the flags2 `BCM_OAM_ENDPOINT_FLAGS2_RDI_CLEAR_ON_LOC_CLEAR`. By default, this feature is disabled.
3. The CCM RDI bit for a local MEP may be determined by the RDI bit of an incoming CCM received at the RMEP. This can be enabled with the flag2 `BCM_OAM_ENDPOINT_FLAGS2_RDI_ON_RX_RDI`. By default, this feature is disabled.
4. Upon an LoC or LoC-clear event, the RMEP can perform the following actions (only one may be selected), which are controlled through `flags2` and `flags`:
   a. `BCM_OAM_ENDPOINT_FLAGS2_REMOTE_UPDATE_STATE_DISABLE` – Do not update the fault state but generate an event. This means events will be generated repeatedly until the CPU updates faults in the RMEP flag through `bcm_oam_endpoint_create()`.
   b. `BCM_OAM_ENDPOINT_REMOTE_EVENT_DISABLE` – Update the fault field but do not generate an event (this is in `flags`, and not `flags2`).
   c. `BCM_OAM_ENDPOINT_RDI_AUTO_UPDATE` – Update the fault fields unconditionally and generate an event. Events may be lost in this case.
   d. If no flag is specified, update the fault state conditionally (if the event FIFO is not full) and generate event.

5. Upon a port/interface state or RDI change detected in the RMEP, the RMEP can perform the following actions (only one may be selected), which is controlled through flags2:
   a. `BCM_OAM_ENDPOINT_FLAGS2_REMOTE_UPDATE_STATE_DISABLE` – Do not update the RDI/state value but generate an event. This means events will be generated repeatedly until the CPU updates the state in the RMEP through `bcm_oam_endpoint_create()`.
   b. `BCM_OAM_ENDPOINT_FLAGS2_RX_REMOTE_EVENT_DISABLE` – Update the state/RDI but do not generate an event.
   c. `BCM_OAM_ENDPOINT_FLAGS2_RX_REMOTE_DEFECT_AUTO_UPDATE` – Update the state/RDI unconditionally and generate an event.
   d. If no flag is specified, update the state/RDI conditionally (if the event FIFO is not full) and generate an event.

In Item 4 and Item 5, the difference between option (c) and (d) is that if the OAMP event FIFO is full, in option (c), the RMEP state will be updated but an event will not be generated. In option (d), the RMEP state will not be updated, until the event FIFO is not full, at which point an event will be generated and the RMEP state will be updated. Options (b) or (c) should be used when relying on protection packet for information on incoming events. Option (d) should be used when relying on interrupts. Option (a) may be used for debugging.

In the local MEP, the following `flags2` may be set:

By default, RDI update in local MEP is enabled for both Received packet and LOC from RMEP. This can be disabled individually by Local endpoint create flags `BCM_OAM_ENDPOINT_FLAGS2_RDI_FROM_RX_DISABLE` and `BCM_OAM_ENDPOINT_FLAGS2_RDI_FROM_LOC_DISABLE`, respectively.

33.15.5 Down MEP Egress Injection

On standard Ethernet Down MEPs, injected PDUs from the OAMP do not go through egress PP processing. This means these packets do not increment LM counters. Alternatively, it is possible to define a *Down MEP egress injection MEP*. For these endpoints, OAM PDUs will be injected with PP system headers such that the packet goes through full PP processing in the egress. This includes the following activities:

- OAM classifier and LM counters – PDUs can increment or not increment the MEP's counters according to the configuration defined on the profile through `bcm_oam_profile_action_set()`. Counters on MEPs in a lower MD-level will be incremented, while maintaining the hierarchical LM limitations.
- Egress VLAN translation (if applicable) –Packet encapsulation in `bcm_oam_endpoint_create()` (`vlan` and `tpid` fields) should be defined as packets on the LIF appear before egress VLAN translation and after ingress VLAN translation. VSI must be provided to `bcm_oam_endpoint_create()` through the `vpn` field.

These endpoints are created with the flag2 `BCM_OAM_ENDPOINT_FLAGS2_EGRESS_INJECTION_DOWN`.

To produce response packets (LMR, DMR, or LBR) where the VLAN tag structure of incoming query packets is the same as what is configured on the MEP, two additional flags are required.

- If incoming query packets have one tag, define the MEP with one VLAN tag and use flags2 `BCM_OAM_ENDPOINT_FLAGS2_EGRESS_INJECTION_DOWN_IVEC_DELETE_ONE_VLAN_TAG` or `BCM_OAM_ENDPOINT_FLAGS2_EGRESS_INJECTION_DOWN` to create the MEP. One VLAN tag from the injected OAM packet will be deleted by the ingress VLAN edit command carried in the PPH_FHEI header. VLAN tags on transmitted packets from the OAMP will then be built according to the egress VLAN edit.
- If incoming query packets have two tags, define the MEP with two VLAN tags and use flags2 `BCM_OAM_ENDPOINT_FLAGS2_EGRESS_INJECTION_DOWN_IVEC_DELETE_TWO_VLAN_TAGS` or `BCM_OAM_ENDPOINT_FLAGS2_EGRESS_INJECTION_DOWN` to create MEP. Two VLAN tags from the injected OAM packet are deleted by the ingress VLAN edit command carried in the PPH_FHEI header. VLAN tags on transmitted packets from the OAMP are then built according to the egress VLAN edit.

- If incoming query packets are untagged, define the MEP without VLAN tags and use flags2 `BCM_OAM_ENDPOINT_FLAGS2_EGRESS_INJECTION_DOWN`.

The VLAN edit commands are created through the `bcm_vlan_translate_action_id_create()` and `bcm_vlan_translate_action_id_set()` APIs, and the command values must be shown to the OAMP. This is done through the OAM control type `bcmOamControlIngressVlanEditClassDeleteOneVlanTag` and `bcmOamControlIngressVlanEditClassDeleteTwoVlanTags` to set the ingress VLAN edit command to delete one or two VLAN tags for the injected OAM packet from down MEP egress injection.

33.15.6 SOC Properties

None

33.15.7 Configuration Flow

Fill a `bcm_oam_endpoint_info_t` struct according to the type of endpoint and then call `bcm_oam_endpoint_create()`. Parameters are described in the following sections.

33.15.7.1 Non-Accelerated Endpoints

- `level` – Endpoint's MD-Level
- `type` – Must be one of `bcmOAMEndpointTypeEthernet`, `bcmOAMEndpointTypeBHHMPLS`, `bcmOAMEndpointTypeBHHPwe`, or `bcmOAMEndpointTypeBHHSection`.
- `group` – The endpoints' group. See the previous section.
- `gport` – LIF on which endpoint is defined.
- `id` – ID will be set after calling `bcm_oam_endpoint_create()`
- `lm_counter_base_id` – See Section 33.16, Loss Measurement. A value of 0 does not mean *no stamping*. This is done by using a profile with the flag `BCM_OAM_PROFILE_LM_TYPE_NONE`.
- `lm_flags`
  - `BCM_OAM_LM_PCP` – Use priority-based LM. When this is set, `lm_counter_base_id` must be a multiple of eight.
- `flags` – The following flags are supported
  - `BCM_OAM_ENDPOINT_UP_FACING` – Define an Up MEP
  - `BCM_OAM_ENDPOINT_INTERMEDIATE` – Define a MIP

33.15.7.2 Accelerated Local Endpoints

Fill a `bcm_oam_endpoint_info_t` structure according to the type of endpoint and then call `bcm_oam_endpoint_create()`. Parameters are:

- `level` – Endpoint's MD-Level
- `type` – Must be one of `bcmOAMEndpointTypeEthernet`, `bcmOAMEndpointTypeBHHMPLS`, `bcmOAMEndpointTypeBHHPweGAL`, `bcmOAMEndpointTypeBHHPwe`, or `bcmOAMEndpointTypeBHHSection`.
- `group` – The endpoints' group. See section above.
- `gport` – LIF on which endpoint is defined.
- `lm_counter_base_id` – See Section 33.16, Loss Measurement.
- `lm_counter_if` – See Section 33.16, Loss Measurement.
- `id` – See MEP-DB management section. May be set with ID or ID may be set by SDK.

- flags: The following flags are supported:
  - BCM_OAM_ENDPOINT_UP_FACING – Define an Up MEP.
  - BCM_OAM_ENDPOINT_REPLACE – Replace fields in the endpoint. Not all fields may be updated.
    NOTE: BCM_OAM_ENDPOINT_REPLACE is used to replace the same type endpoint fields. For updating different type endpoint fields, destroy the endpoint first, then create a new one.
  - BCM_OAM_ENDPOINT_WITH_ID
  - BCM_OAM_ENDPOINT_RDI_TX – Manually set the RDI bit on outgoing CCMs
- opcode_flags
  - BCM_OAM_OPCODE_CCM_IN_HW – This flag indicates that the endpoint is accelerated
- tx_gport, int_pri, egress_label, src_mac_address, outer_tpid, inner_tpid, vlan, inner_vlan, pkt_pri, inner_pkt_pri – See Section 33.15.2, Endpoint Types and Encapsulation
- port_state, interface_state – See section above.
- mpls_out_gport – Must be set on endpoints of type MPLS/PWE when LM counting is required. Counter on the egress is set on this port. When lm_counter_base_id is set to 0, there is no point in setting this field as well.
- ccm_period – Must use one of the BCM_OAM_ENDPOINT_CCM_PERIOD_* defines.
- acc_profile_id – See Section 33.4, OAM Profiles and Classification.
- name – MEG ID on outgoing CCMs.
- flags2
  - BCM_OAM_ENDPOINT_FLAGS2_RDI_FROM_RX_DISABLE – See above
  - BCM_OAM_ENDPOINT_FLAGS2_RDI_FROM_LOC_DISABLE – See above
  - BCM_OAM_ENDPOINT_FLAGS2_EGRESS_INJECTION_DOWN – See above
  - BCM_OAM_ENDPOINT_FLAGS2_SIGNAL – See Section 33.24, Signal Degrade Indication.
- rx_signal, tx_signal – See Section 33.24, Signal Degrade Indication.
- dst_mac_address – Optional field to support unicast DA transmission.

33.15.7.3 Accelerated Remote Endpoints

Fill a bcm_oam_endpoint_info_t struct according to the type of endpoint and then call bcm_oam_endpoint_create(). Parameters are:

- local_id ID of the RMEP's local endpoint
- name – MEG-ID on incoming CCMs. Classification will be based on this field.
- ccm_period – Expected incoming CCM LOC detection time (in milliseconds). By default 3.5 * local endpoint's period.
- id – See MEP-DB management section. May be set with ID or ID may be set by SDK.
- flags
  - BCM_OAM_ENDPOINT_REPLACE – Replace fields in the endpoint. Not all fields may be updated.
  - BCM_OAM_ENDPOINT_WITH_ID
- loc_clear_threshold – Number of packets required to reset the Loss-of-Continuity status. Setting to 0 disables the automatic clearing of LOC indication and may not be greater than 3.
- faults
  - BCM_OAM_ENDPOINT_FAULT_REMOTE – Last CCM was received with RDI (read only flag)
  - BCM_OAM_ENDPOINT_FAULT_CCM_TIMEOUT – The flag has different meanings on read on write.
    - Read: The endpoint is in a timeout state.
    - Write: Create the endpoint directly time-in state. (Otherwise, LoC may occur only after the first CCM arrived. See Section 33.15.1, Local and Remote Endpoints.)
- flags2 – See the explanation above
  - BCM_OAM_ENDPOINT_FLAGS2_REMOTE_UPDATE_STATE_DISABLE
  - BCM_OAM_ENDPOINT_FLAGS2_RX_REMOTE_EVENT_DISABLE

  - BCM_OAM_ENDPOINT_FLAGS2_RX_REMOTE_DEFECT_AUTO_UPDATE
  - BCM_OAM_ENDPOINT_FLAGS2_REMOTE_UPDATE_STATE_DISABLE
  - BCM_OAM_ENDPOINT_REMOTE_EVENT_DISABLE
  - BCM_OAM_ENDPOINT_RDI_AUTO_UPDATE
  - BCM_OAM_ENDPOINT_FLAGS2_RDI_ON_LOC
  - BCM_OAM_ENDPOINT_FLAGS2_RDI_CLEAR_ON_LOC_CLEAR
  - BCM_OAM_ENDPOINT_FLAGS2_RDI_ON_RX_RDI
- sampling_ratio – Rate of punt packets transmitted per state-change event
  - 0 – No packets sampled to the CPU
  - 1 to 8 – Count of packets (with events that need to arrive before one is sampled to the CPU)
    - 1 – Punt every state change event
    - 8 – Punt 1 of every 8 events

33.15.8 Shell Commands

The oam oamEP command displays all created endpoints.

33.15.9 Application Reference

Refer to cint_dnx_oam.c for basic examples.

For Down MEP egress injection, refer to cint_oam_egress_injection.c.

33.16 Loss Measurement

Loss measurement is used to collect counter values applicable for ingress and egress service frames where the counters maintain a count of transmitted and received data frames between a pair of MEPs.

33.16.1 Counter Resource Management

Counter resource management is discussed in the *Traffic Manager Programming Guide*.

Counter resources in the BCM88690 are managed using counter databases which are mapped to counter engines.

Counters are accessed using stat commands, each one having a command_id.

OAM counter resources are managed by OAM dedicated counter databases. OAM dedicated counter databases, link OAM counter_if to a specific stat_command_id.

OAM counting is performed using command_id 7,8,9 in ingress, 0,1,2 in egress. Therefore OAM databases should be defined using these command_ids.

Example cint: set_counter_resource() in cint_oam_basic.c

33.16.2 Endpoint Counter Assignment

Counters may be assigned to count an endpoint using bcm_oam_endpoint_create().

Endpoints without counter should always have lm_counter_base_id = 0. Lm-endpoints should always have lm_counter_base_id > 0.

When a counter is assigned to an endpoint, additional counter with the same `counter_if` and same `counter_base` is assigned on the opposite side, except for ingress only and egress only endpoints. For that reason counter resources should be allocated for both ingress and egress side.

33.16.3 Hierarchical Loss Measurement

Hierarchical loss measurement (HLM) refers to the ability to count one packet on more than one counter.

Counters that needed to be incremented by the same packet must reside on different counter engines. This may be applied by assigning different counter databases to different `counter_if`.

HLM-counter assignment rules:

- HLM by level (Ethernet OAM):
  - Up to three endpoints with valid counters may be assigned on the same LIF

    Additional endpoints on the same LIF may be assigned without counters.
  - If one endpoint with a counter resides on a LIF, it can be either the endpoint with the highest MD-Level or the lowest MD-Level (in other words, on LIF X, it is possible to configure an endpoint at MD-Level 5 with a counter and endpoints with MD-Level 3, 2, and 1 without a counter).
  - If two endpoints with a counter reside on a LIF, they can only be the endpoints with the highest MD-Level and the lowest MD-Level.
  - If three endpoints with a counter reside on a LIF, those with counters may only be the endpoints with the highest MD-Level and the two lowest MD-Levels.
  - Only a single endpoint may be configured on the MPLS LIF.
  - Counters associated with endpoints on the same LIF should reside on different counter `counter_if` (0,1,2). Each of the `counter_if` should be mapped to a different counter engine.

  Endpoints without counter:
  - Endpoints without a counter should have c`ounter_base=0`.
  - All endpoints without a counter assigned should have the same `counter_if`.
  - If three endpoints with a valid counter reside on a LIF, all endpoints without a valid counter may have any `counter_if`.
  - Otherwise, endpoints without a valid counter must have a `counter_if` different than other endpoints with a valid counter.
- HLM by LIF (MPLS/PWE/Section OAM):
  - Counters may be configured on endpoints which reside on up to 3 LIFs of the same expected packet.
  - Counters associated with endpoints on the same expected packet, should reside on different `counter_if`, each of them should be mapped to a different counter engine.

Examples:

For Ethernet OAM, the consider the below, valid, calling sequence:

1. Configure endpoint A with `level` 7, `gport` 0x1000, `lm_counter_base_id` 10 ,`lm_counter_if` 0.
2. Configure endpoint B with `level` 4, `gport` 0x1000, `lm_counter_base_id` 20, `lm_counter_if` 1.
3. Configure endpoint C with `level` 3, `gport` 0x1000, `lm_counter_base_id` 30, `lm_counter_if` 2.
4. Configure endpoint D with `level` 5, `gport` 0x1000, `lm_counter_base_id` 0, `lm_counter_if` 1.
5. Configure endpoint E with `level` 6, `gport` 0x1000, `lm_counter_base_id` 0, `lm_counter_if` 1.
6. Destroy endpoint B.

NOTE:

- If Step 3 and Step 4 are swapped, the sequence results in an error because endpoint D, which does not have a valid counter, shares the same `counter_if` with endpoint B, which does have a valid counter. At that stage, only two endpoints with counters are configured.
- If endpoint E is given MD-Level 2, this results in an error because valid counters on LIF 0x1000 are no longer at the endpoints (the highest MD-Level and 2 lowest MD-Levels).
- If endpoint A is destroyed in Step 6, instead of endpoint B, the sequence is incorrect because endpoints D,E, and B share the same `counter_if`, even though endpoint B has a valid counter while D and E do not.
- Hierarchical LM is not available in BCM88670 system headers mode (IOP).

33.16.4 Priority-Based Counting

Priority (PCP) based counting functionality enables counting packets on different counters by their priority.

PCP based counting rules:

1. Eight sequential counters should be assigned on each endpoint that use PCP based counting.
2. Counter assignment is done using the counter for PCP = 0 (`lm_counter_base`).
3. `lm_counter_base` should be a multiple of eight.
4. All PCP based counters should have values outside the `bcmOamControlLmPcpCounterRangeMin`, `bcmOamControlLmPcpCounterRangeMax`.
5. `lm_flag BCM_OAM_LM_PCP` should be used during `bcm_oam_endpoint_create()`

Examples:

In `cint_oam_basic.c`:

- `oam_pcp_based_counting()`
- `oam_upmep_pcp_example()`
- `oam_mpls_pcp_based_counting()`

The OAM-PCP within the pipeline can be calculated in the following modes:

- At the ingress, take the value from the packet. For Ethernet OAM, this will be the PCP on the innermost VLAN. For MPLS/PWE packets, this will be taken from the EXP.
- At the egress, take the value from the TC, taken from ingress system headers

33.16.4.1 LM Types

LM can be performed in three ways:

- Single-ended LM
- Dual-ended LM
- None (no LM stamping performed in the classifier)

Type of the LM to be performed should be applied using action_set, described in Section 33.4, OAM Profiles and Classification.

Example: In `cint_oam_action.c`: `oam_set_of_action_profiles_create()`

LM stamping for ingress multicast packets is not supported.

33.16.5 SOC Properties

None

33.16.6 Configuration Flow

- `bcm_oam_control_set()` is used to configure `bcmOamControlLmPcpCounterRangeMin`, `bcmOamControlLmPcpCounterRangeMax`
- Counter Resource Allocation – CRPS

33.16.7 Shell Commands

The `OAM OAMEndPoint` command shows endpoints with LM parameters.

The `oam coUnter` command displays counter per endpoint.

33.16.8 Application Reference

- `set_counter_resource()`
- `oam_pcp_based_counting()`
- `oam_mpls_pcp_based_counting()`
- `oam_upmep_pcp_example()`
- `oam_set_of_action_profiles_create()`

33.17 Synthetic Loss Measurement

Synthetic loss measurement collects counter values applicable for OAM service frames where the counters maintain a count of transmitted and received SLM and SLR frames between a pair of MEPs on a specific interval.

This section describes how to set up and manage an SLM session in the OAMP.

Non-accelerated SLM is not available. For SLM sessions managed by external hardware, counting and stamping should be managed by the external HW.

33.17.1 Counter Resource Management

Counter resource management is discussed in the *Traffic Manager Programming Guide*. Counter resources in the BCM88690 are managed using counter databases that are mapped to counter engines. Counters are accessed using stat commands, and each command has a `command_id`.

OAM counter resources are managed by OAM-dedicated counter databases. OAM-dedicated counter databases link OAM `counter_if` to a specific `stat_command_id`. OAMP counting is performed using `command_id` 0. Therefore, OAM databases should be defined using this `command_id`.

Example: cint: `set_counter_resource()` in `cint_oam_basic.c`

33.17.2 Counter Assignment

Counter assignment is done through `bcm_oam_loss_add()`. For this purpose the endpoints selected for SLM measurement should have always been created with `lm_counter_base_id = 0`.

In addition, OAM profiles the SLM sessions use should be created with the flag `BCM_OAM_PROFILE_LM_TYPE_NONE`.

The type of the LM to be performed should be applied using `action_set`, described in Section 33.4, OAM Profiles and Classification.

Example: In `cint_oam_action.c`: `oam_set_of_action_profiles_create()`

33.17.3 Initiator and Responder

Because each endpoint can maintain only two counters, the single endpoint may act as initiator or responder. The initiator sends SLM frames, and the responder replies to incoming SLMs with SLRs.

Unlike in dual-ended or single-ended LM (CCM or LMM/R based), for the synthetic case, it is necessary to call `bcm_oam_loss_add()` for the responder role as well as the initiator.

Both endpoints should have valid counter assignments, but only the initiator should have a valid transmit rate.

NOTE: The SLM packets cannot be processed in the OAMP without a valid SLM session.

33.17.4 SLR Measurement

In the BCM88830, the measurement of all incoming valid SLRs is recorded.

In the BCM88690, BCM88800, and BCM88480 devices, the measurement is recorded only after `bcm_oam_loss_add()` is called with the `BCM_OAM_LOSS_UPDATE_NEXT_RECEIVED_SLR` flag for the given endpoint.

33.17.5 Measurement Period

In the BCM88690, BCM88800, and BCM88480 devices, if `bcmOamControlEnableNextReceivedSlr` is enabled through `bcm_oam_control_set()`, SLM can only be added on OAM endpoints with `memory_type bcmOamEndpointMemoryTypeLmDmOffloadedEntry`. In this case, statistics are recorded on all SLRs received at the OAMP, and only after ~2 minutes after device bring up.

33.17.6 Configuration Flow

33.17.6.1 Allocating a Counter

Allocate OAMP counters.

OAMP counters should be allocated with `eviction=0`.

Example: In `cint_oam_statistics.c`: `cint_oam_oamp_counter_allocation_main()`

33.17.6.2 Creating an Endpoint

Call `bcm_oam_endpoint_create()`. Mandatory parameters are:

- `lm_counter_base_id` – Should be with value of 0
- `lm_counter_if` – Should be with value of 0
- `acc_profile_id` – Must be a profile created with the flag `BCM_OAM_PROFILE_LM_TYPE_NONE`

The following limitations exist:

- For the BCM88690 (Jericho2) and the BCM88480 (Q2A), the SLM entry cannot be added to an EP with the ID placed in bank 0 to bank 3.

33.17.6.3 Adding Loss

Call `bcm_oam_loss_add()`. Mandatory parameters are:

- `measurement_period` – Set to 0.
- `slm_counter_core_id` – Select the CRPS core
- `slm_counter_base_id` – Select the counter pointer which SLMs will increment
  - SLR will increment the counter `slm_counter_base_id + 1`
- `flags`
  - `BCM_OAM_LOSS_SLM`

33.17.6.4 Controlled Measurement

Manual measurement can perform measurement by request. For this purpose call `bcm_oam_loss_add()` :

- `flags` – `BCM_OAM_LOSS_UPDATE_NEXT_RECEIVED_SLR`
- `id` – Provide endpoint id in case of offloaded endpoints.
- `loss_id` – Provide loss ID for self-contained endpoints.

33.17.6.5 Configuring an OAMP Destination for SLM Packets

Example: In `cint_oam_actions.c`: `oam_set_opcode_destination_to_oamp()`

33.17.7 Shell Commands

None

33.17.8 Application Reference

Use the following application references:

- In `cint_oam_statistics.c`: `cint_oam_oamp_counter_allocation_main()`
- In `cint_oam_performance_measurement: dnx_oam_slm_run_test()`
- In `cint_oam_actions.c`: `oam_set_opcode_destination_to_oamp()`
- In `cint_utils_oam.c`: `enable_oam_measure_next_slr()`

33.18 Trap and Snoop Information

This section describes how trapped and snooped OAM packets are presented at the CPU.

33.18.1 Ingress Trapped Packet Structure

Ingress trapped packets contain only one set of system headers.

- Trapped OAM ingress packets arrive with the trap codes `bcmRxTrapOamLevel`, `bcmRxTrapBfdOamDownMEP`, and `bcmRxTrapOamPassive`.
- Snooped packets arrive with the trap code `bcmRxTrapSnoopOamPacket`. In certain cases, these packets may appear without a trap code. In such cases, use the API `bcm_mirror_header_info_set()`.

33.18.2 Trapped DM Packets

In a trapped DM packet for non-accelerated endpoints, a 32-bit TOD second is inserted between the system header and the network header. Account for the wraparound using the low 34-bit TOD in the ASE header, which include bits for seconds.

The existence of these 4 bytes can be determined according to the sub-type in the ASE. Values of 2 or 3 indicate a DM, which means these 4 bytes appear.

33.18.3 Trap Code Qualifier

The trap code field indicates the trap code. The trap-code-qualifier is encoded in the following way:

- For accelerated endpoints, trap-code-qualifier is {1'b0, mp_profile[1:0], oam_id}
- For non-accelerated endpoints, trap-code-qualifier is {1'b1, 2'b0, 1'b1, my_cfm_mac(1), ingress(1), packet_is_bfd(1), oam_opcode(4), internal-mp-type-encoding(4), mp_profile(4)}

33.18.4 SOC Properties

None

33.18.5 Configuration Flow

None

33.18.6 Shell Commands

None

33.18.7 Application Reference

None

33.19 OAM Identification TCAM

Some OAM applications require additional TCAM entries. This bank is shared with the field processor.

The following applications consume TCAM entries:

- `bcmOamControlEthDmacMdlMatchCheck` – 42 entries
- `bcmOamControlEnableBfdPhp` – One entry
- `bcmOamControlMplsOamMdlIgnore` – 14 entries
- `bcmOamControlNativeEthernetOverPweClassification` – One entry

33.19.1 Soc Properties

None

33.19.2 Configuration Flow

Call `bcm_field_tcam_bank_add` with `bcmFieldAppDbOamIdentification` to create TCAM BANK for OAM identification.

Specific applications are added through `bcm_oam_control()`.

33.19.3 Shell Commands

Use `dbal table dump table=OAM_TCAM_IDENTIFICATION_DB` to show the allocated entries.

33.20 OAMP Per-Endpoint Statistics

The OAMP statistics count OAM packets received and transmitted by the OAMP.

In the BCM88690, OAMP has direct access to CRPS. This enables OAMP to provide statistics functionality per endpoint. The user may configure the OAMP to count the following items:

- Packets to and from specific MEPs
- RX packet, TX packets, or both
- Packets with specific OpCodes

For each packet received or transmitted by the OAMP, a counter pointer is calculated. Counting of a specific packet type may be enabled depending on its direction (Tx/Rx) and OpCode. Per OpCode counting may be configured.

The `counter_id` is calculated by the following formula, and the user must configure the range according to the formula. The meaning of each item is as follows:

- `tx_counter_id` = MEP-ID << Mep-Profile.Mep-Id-Shift + Opcode-For-Count-Index
- `rx_counter_id` = 1 << CFG_RX_SHIFT + MEP-ID << Mep-Profile.Mep-Id-Shift + Opcode-For-Count-Index

33.20.1 Counting Packets with Specific OpCodes

In the BCM88690, OAMP can count different OpCodes on separate counters. For this purpose, the specific OpCode counting must be allowed.

The counting per OpCode may be configured globally through the `bcm_oam_control_indexed_set()` API:

- Set the type to be `bcmOamControlOampStatsTxOpcode` or `bcmOamControlOampStatsRxOpcode`
- Use `value` to determine whether to count. A value of 0 disables the counting, and a value of 1 enables the counting.
- Set the `index` with the actual OpCode

NOTE: Each OpCode must be enabled or disabled per direction.

NOTE: RX LMM and TX LMR are taken as one action, and they are counted only once on the RX LMM counter. The same rule applies for all replay packets.

The following table shows how to calculate the `counter_id`.

Actual Packet OpCode	OpCode for Count Index
CCM/BFD	0
LMM/SLM/1SL/RFC6374-LM-Query	1
LMR/SLR/RFC6374-LM-Response	2
DMM/1DM/RFC6374-DM-Query	3
DMR/RFC6374-DM-Response	4
Other	5

Example: `cint_oam_oamp_statistics_main()` in `cint_oam_statistics.c`

33.20.2 Counter Resource Management

Counter resource management is discussed in the *Traffic Manager Programming Guide*. Counter resources in the BCM88690 are managed using counter databases that are mapped to counter engines. Counters are accessed using stat commands, each one having a `command_id`.

OAM counter resources are managed by OAM-dedicated counter databases. OAM-dedicated counter databases, link OAM `counter_if` to a specific `stat_command_id`. OAMP counting is performed using `command_id` 0. Therefore, OAM databases should be defined using `command_id`.

Example: cint: `cint_oam_oamp_statistics_main()` in `cint_oam_statistics.c`

33.20.3 Configuring PREFIX and SHIFT Values per Direction

The value for `CFG_RX_SHIFT` may be configured globally through the `bcm_oam_control_set()` API.

- Set the type to be `bcmOamControlOampStatsShift`
- Use `value` to set the SHIFT value.

  Example: cint : `cint_oam_oamp_statistics_main()` in `cint_oam_statistics.c`.

33.20.4 Endpoint Create with OAMP Statistics

To enable the statistics feature per Endpoint, create an accelerated MEP and one of the statistics `flags2`:

- `BCM_OAM_ENDPOINT_FLAGS2_TX_STATISTICS`
- `BCM_OAM_ENDPOINT_FLAGS2_RX_STATISTICS`

It is recommended to create the endpoint with the `WITH_ID` flag. The Endpoint ID should be in a range based on allocated counter IDs.

The `BCM_OAM_ENDPOINT_FLAGS2_PER_OPCODE_STATISTICS` flag may be used to enable the per-OpCode count feature. When this flag is set, at least one of the other two must be set as well. If this flag is set, then Mep-Profile.Mep-Id-Shift will be 3.

33.20.5 SOC Properties

None

33.20.6 Configuration Flow

1. Enable counting per OpCode:

```
key.type = CCM_OPCODE;
value =1;
bcm_oam_control_indexed_set(unit, bcm_switch_control_key_t key, value);
```

2. Define `endpoint_id` range and allocate needed counters.

```
cint_oam_oamp_statistics_main(unit, 32768, 1024, OAM_STAT_COUNT_PER_DIRECTION)
```

3. Call `bcm_oam_endpoint_create()`

```
bcm_oam_endpoint_info_t_init(&mep_acc_info);
mep_acc_info.id=32768;
mep_acc_info.flags2|= BCM_OAM_ENDPOINT_FLAGS2_TX_STATISTICS |
BCM_OAM_ENDPOINT_FLAGS2_RX_STATISTICS;
…
/*mep_acc_info_config*/
...
rv = bcm_oam_endpoint_create(unit, &mep_acc_info);
```

33.20.7 Shell Commands

None

33.20.8 Application Reference

`cint_oam_oamp_statistics_main()` in `cint_oam_statistics.c`:

`cint_oam_run_with_defaults()` in `cint_sand_oam.c`.

33.20.9 OAMP Events and Interrupts with DMA

OAM events and reports are stored in an event FIFO. The system can wait for event interrupts and then invoke user call backs. An interrupt is raised as soon as events are pushed into the FIFO. Alternately, you can set a background DMA-FIFO process that clears the internal FIFO into a user FIFO within the host. An interrupt is triggered according to the schemes:

- When user FIFO is not empty and a time out counter expired since last interrupt.
- When user FIFO has reached some threshold of accumulated reads from the OAMP event FIFO.

Once a DMA interrupt occurs, the software reads as many entries in the host memory as the DMA wrote, parses them, and then calls a user-callback for each unique event.

The device maintains two interfaces each with a FIFO-DMA process:

- High priority for Continuity event (Event interface) and LM/DM threshold violation.
- Low priority for LM/DM reports.

The advantage of using the DMA as opposed to standard flow is that the DMA may be able to handle a larger burst of events. Additionally it is possible to cluster multiple events together with the DMA. Performance may be better using the DMA when the threshold is set to 32 and greater.

However even with the DMA SOC properties set, if enough events are triggered for enough remote MEPs at once, events can be produced so quickly by the OAMP that the DMA will not be able to catch up, and events will be lost.

33.20.10 SOC Properties

To configure the DMA FIFO process for OAM events:

1. Set SOC property `oamp_fifo_dma_event_interface_enable` to enable the FIFO-DMA process. Default is 0.
2. Set SOC property `oamp_fifo_dma_event_interface_buffer_size` to configure the size in bytes of the user FIFO. It should be greater than `oamp_fifo_dma_threshold` × 80B, and smaller than 16384 × 80B.
3. Set SOC property `oamp_fifo_dma_event_interface_timout` to configure the time-out interrupt in microseconds when the FIFO is not full. Value 0 disables the time-based interrupt.
4. Set SOC property `oamp_fifo_dma_event_interface_threshold` – The number of writes by the DMA until an interrupt is triggered.

To configure DMA FIFO for statistics reports repeat steps 1 through 4 with the word *report* replacing the word *event*.

33.20.11 Configuration Flow

When SOC properties are set, behavior with and without DMA is identical except for performance and event clustering.

33.20.12 Shell Commands

None

33.20.13 Application Reference

None

33.21 OAM Server/Client

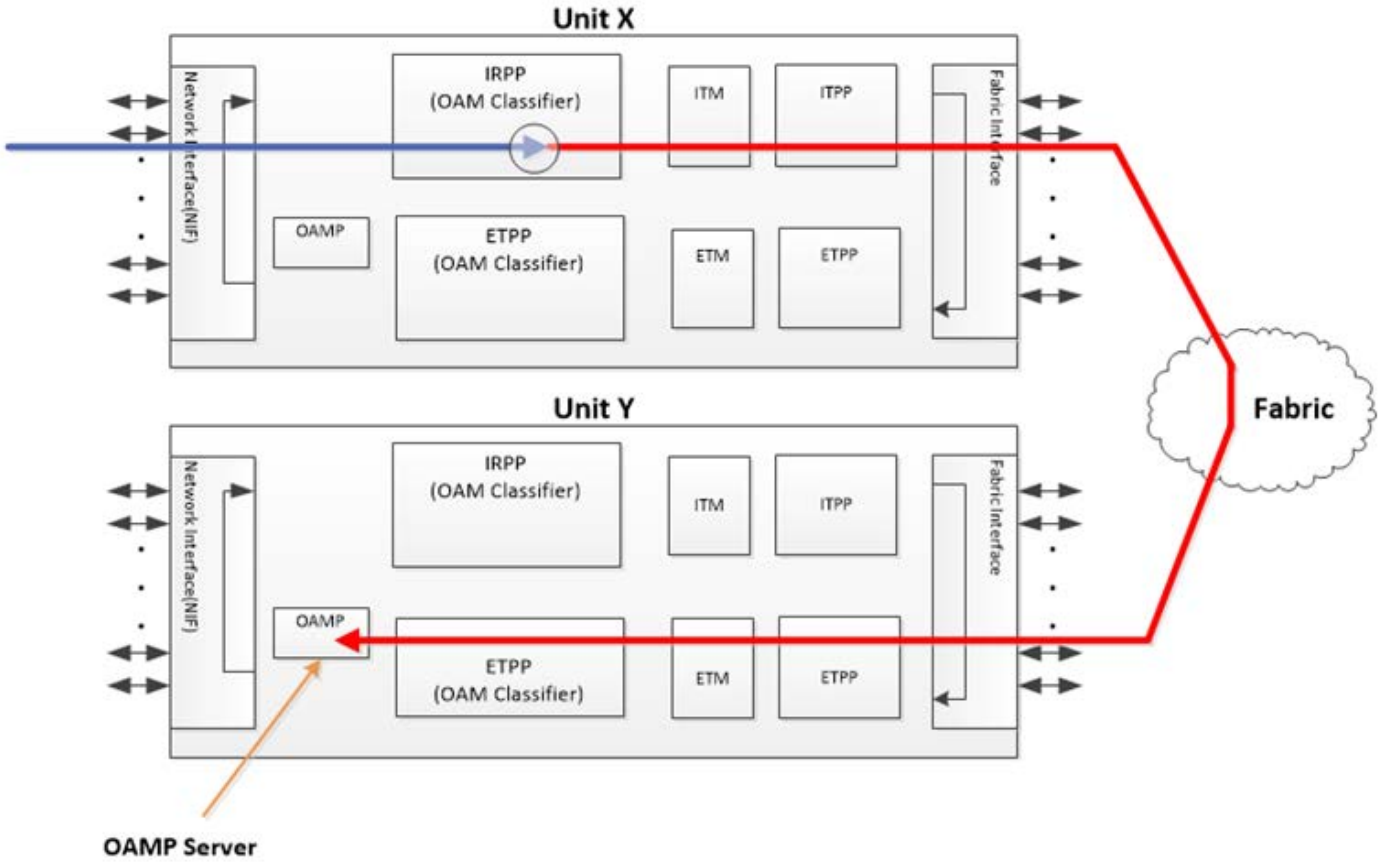
It is possible to create accelerated OAM endpoints on a BCM88690 device using the OAMP on a different BCM88690 device. This configuration is called server/client accelerated OAM, where the OAMP on the server device is used for endpoints on the client device. Transmitted packets originate in the OAMP of the server device, and then are sent to the client device and from there to the remote device (the target). Endpoints are received on the client device and trapped to the OAMP of the server device.

Loss measurement on LAG (that is, multiple client endpoints) is not supported since LM counters are per port and are not synchronized across multiple ports.

Delay measurement TOD must be synchronized between units to correctly support DM in the client/server model.

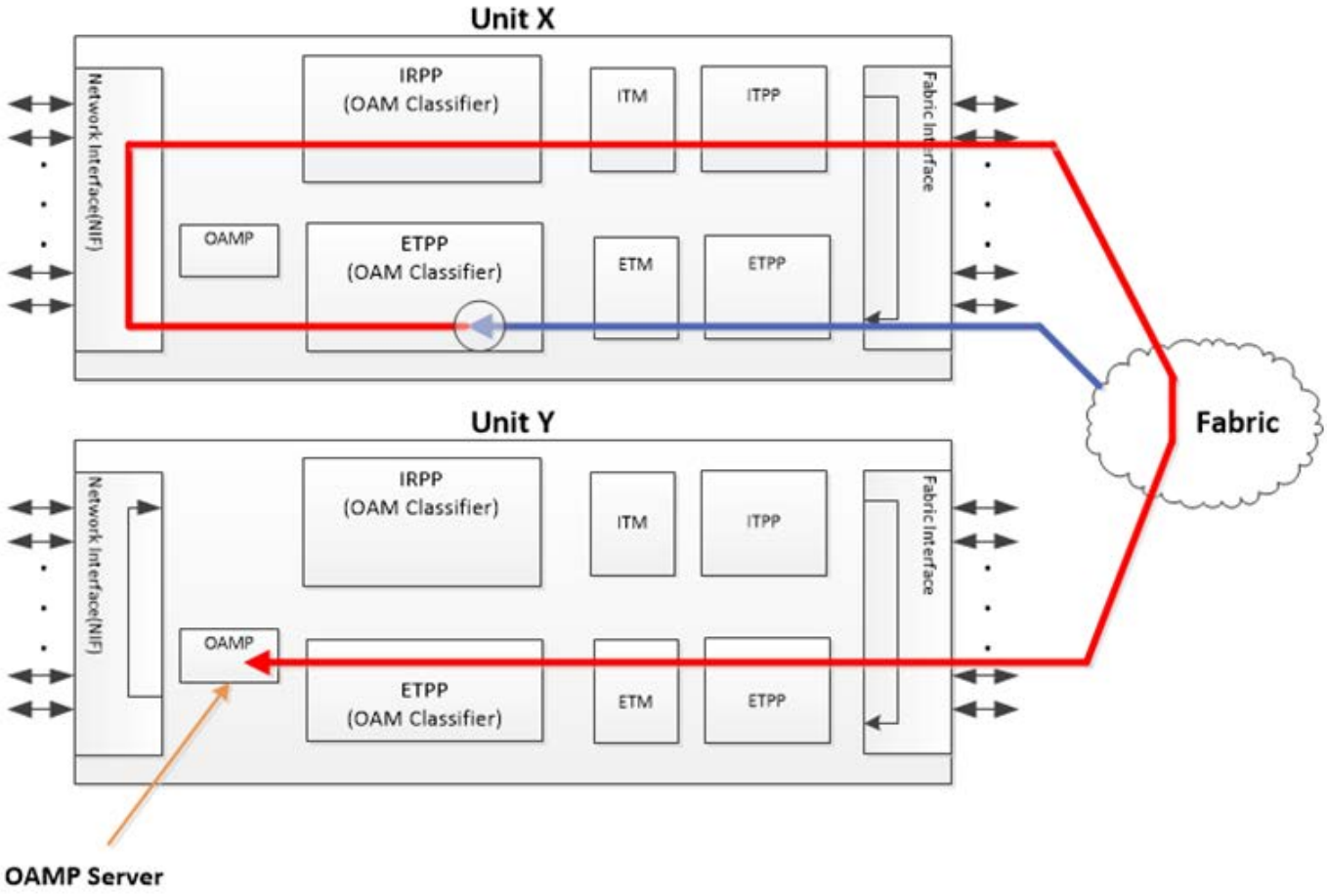
For server/client down-MEPs, received OAM packets are classified at the client's ingress and trapped to the server's OAMP.

Figure 19: OAMP Server Down-MEP Trapping



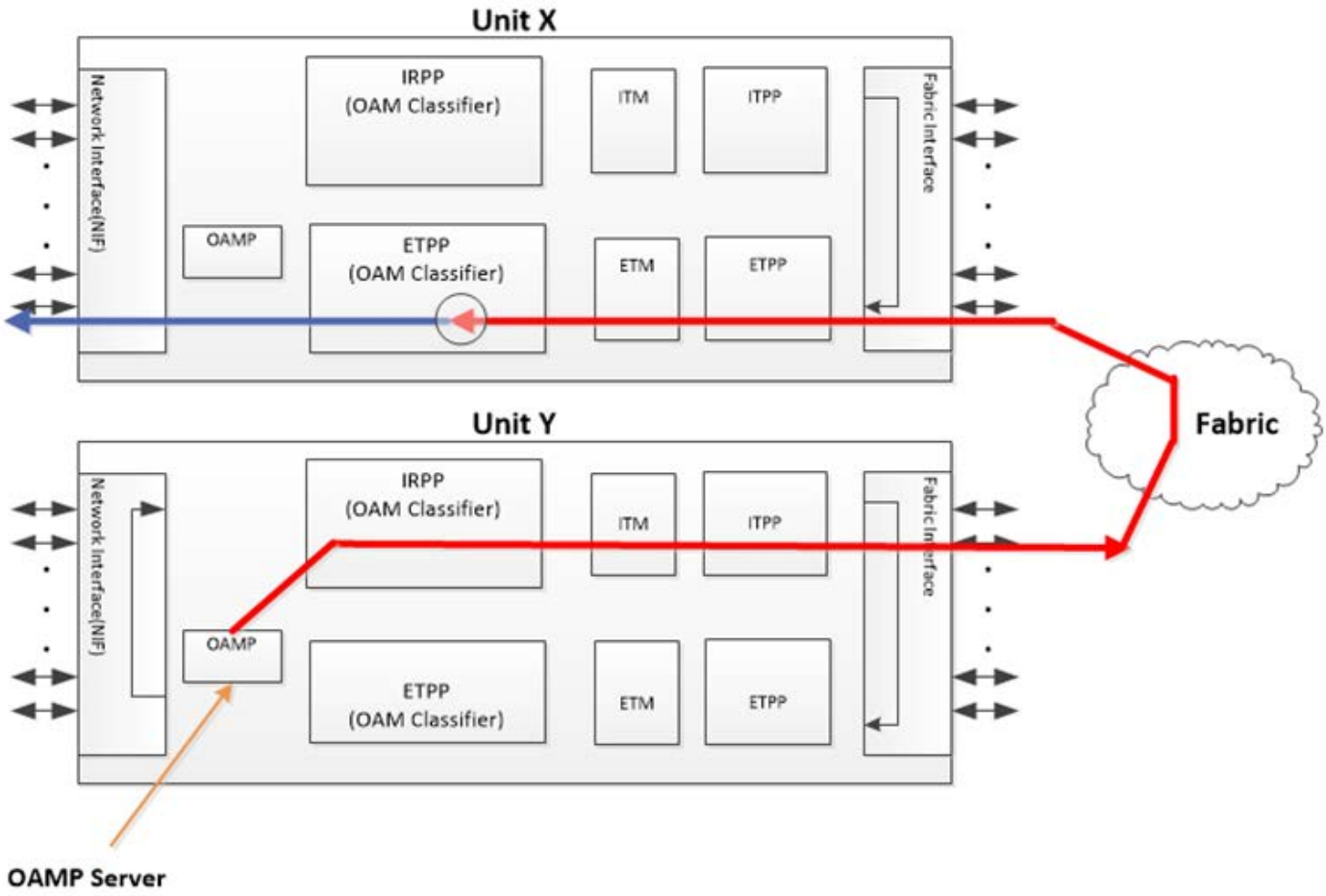
For server/client up-MEPs, received packets are classified in the client's egress, recycled, and trapped to the server's OAMP.

Figure 20: OAMP Server UP-MEP Trapping



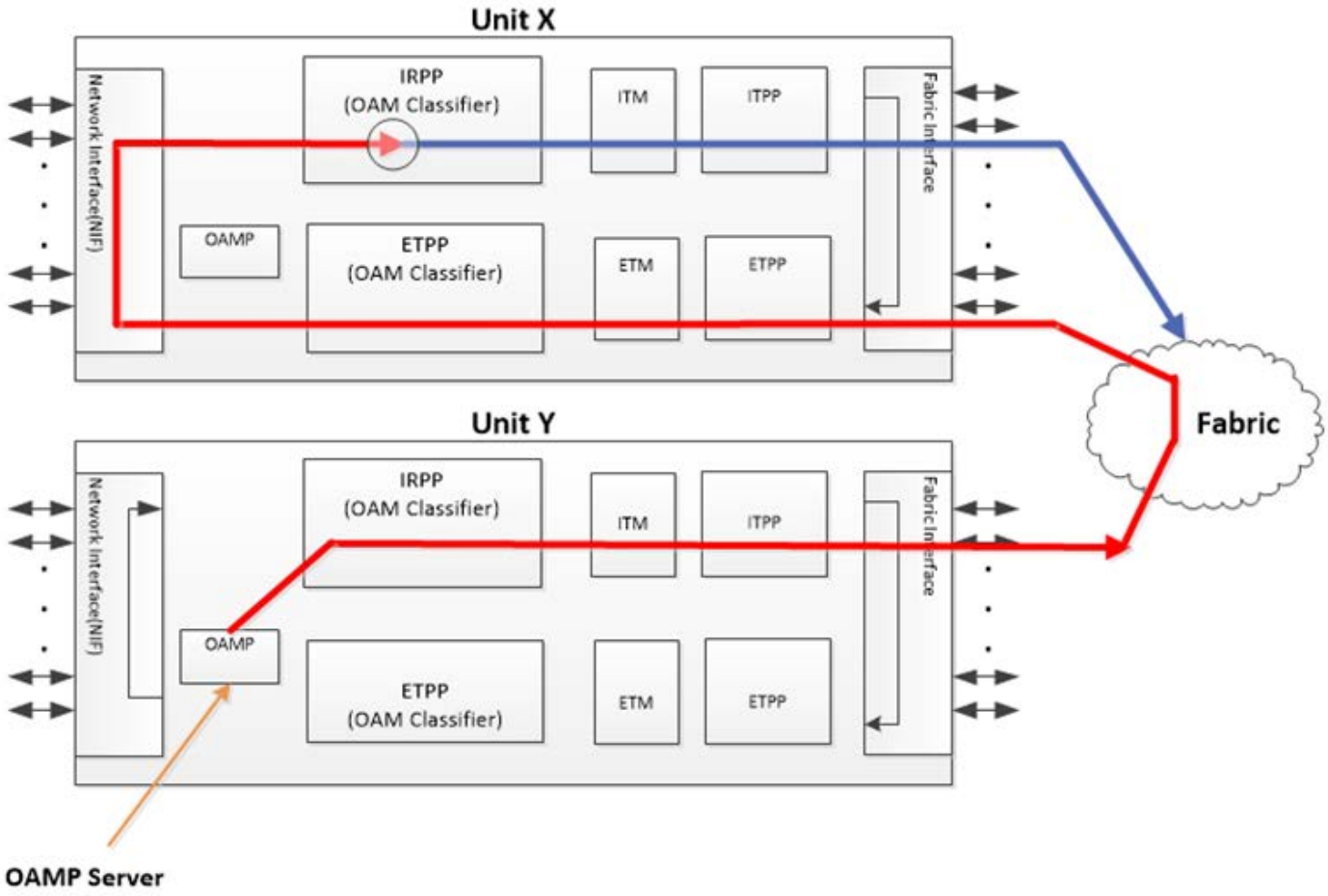
Transmitted packets originate in the server's OAMP. Down-MEPs are forwarded to the clients egress.

Figure 21: OAMP Server Down-MEP Injection



For up-MEPs, transmitted packets are recycled in the client's egress before being sent.

Figure 22: OAMP Server UP-MEP Injection



33.21.1 SOC Properties

None

33.21.2 Configuration Flow

The two main configurations for server/client accelerated endpoints are down-MEP and up-MEP.

33.21.2.1 Down-MEP

The down-MEP configuration flow is as follows:

1. On the server, use `bcm_stk_gport_sysport_get` with `gport=232` (OAMP port) to create a destination. This destination receives remote packets forwarded from the client.
2. On the client, use `bcm_stk_gport_sysport_get` with an ingress port to create a destination. This destination receives packets generated by the server OAMP before sending them to the remote device.
3. On the client, create a user-defined trap:
   a. Use `bcm_rx_trap_type_create` to create the trap code using the parameter `bcmRxTrapUserDefine`.
      - Assuming trapping is required from the multiple client units to a single server endpoint, all traps must be created with the same `trap_code`.
        The first trap should be created normally.
        Next traps (on other units) should be created with the `BCM_RX_TRAP_WITH_ID` flag and the same `trap_code` as the first trap.
   b. Use `bcm_rx_trap_set` to create the trap:
      - Use the trap code defined above.
      - For the field `dest_port`, use the destination created on the server.
      - For the field flags, set `BCM_RX_TRAP_UPDATE_DEST` and `BCM_RX_TRAP_TRAP`.
      - When not using inter-op mode (backwards compatibility to older devices) also set flag `BCM_RX_TRAP_UPDATE_EGRESS_FWD_INDEX` and set field `egress_forwarding_index=7`.
      - When using inter-op mode, also set flag `BCM_RX_TRAP_UPDATE_FORWARDING_HEADER` and set field `forwarding_header=1`
4. On the server, create a down-MEP on this device as you would a regular accelerated down-MEP, with the following exceptions:
   – `remote_gport` is set by using `BCM_GPORT_TRAP_SET` with the above trap code, trap strength 7 and snoop strength 0. When `remote_gport` is valid, the endpoint is treated as a server endpoint.
   – Set gport and `acc_profile_id` as invalid (`BCM_GPORT_INVALID = -1`)
   – Set `tx_gport` to be the system port defined on the client ingress (see above)

NOTE: Aany MEG on the server device can be used, including ones that have non-server endpoints associated with them.

5. On the client, create regular OAM ingress and egress profiles for non-accelerated endpoints.
6. On the client, create an accelerated ingress profile.
7. On the client, create an action using `bcm_oam_profile_action_set` using the following parameters:
   – `flags` = 0
   – `profile_id` – use accelerated ingress profile above
   – `oam_action_key`: set the fields with the following values:
      - `dst_mac_type=bcmOAMDestMacTypeMcast`
      - `endpoint_type=bcmOAMMatchTypeMEP`
      - `flags=0`
      - `inject=0`
      - `opcode=bcmOamOpcodeCCM`
   – `oam_action_result`: set the fields with the following values:
      - `counter_increment=0`
      - `destination`: set with `BCM_GPORT_TRAP_SET` using the above trap, trap strength 7 and snoop strength 0.

- flags=0
- meter_enable=0

8. On the client, create an endpoint on this device as you would a non-accelerated down-MEP with the following exceptions:
   – Set the flag BCM_OAM_ENDPOINT2_HW_ACCELERATION_SET on flags2. When this flag is set, the endpoint is treated as a client endpoint.
   – id – Must be the same ID as the server endpoint's ID.
   – flags – Set flag BCM_OAM_ENDPOINT_WITH_ID.
   – local_id – (Optional) The remote device unit index where the server endpoint is located. This exception is required when multiple endpoints in one unit are client endpoints to server endpoints from different units, but the endpoint ID in the server is the same in each unit.
   – group – Will not be used, but must be valid for verification.
   – level – Must be the same value as in server endpoint.
   – acc_profile_id – Should be the profile configured previously.

NOTE: gport should be set up like a regular endpoint to make sure the client device classifies incoming packets as OAM. This means a valid VLAN GPORT must be created, and ingress and egress OAM profiles (created above) must be associated with it.

33.21.2.2 Up-MEP

On the server, use bcm_stk_gport_sysport_get with gport=232 (OAMP port) to create a destination. This destination will be used to receive remote packets forwarded from the client. (This is the same destination as in down-MEP. If a destination was already created for down-MEPs, it can be used for up-MEPs as well.)

On the client, use bcm_stk_gport_sysport_get with a recycle port to create a destination. This destination will be used to receive packets generated by the server OAMP before sending them to the remote device, as in regular accelerated up-MEPs, after editing the packet will be recycled before being processed and sent to the remote device.

On the client, create two new traps:

1. A special ingress trap that has an editing rule that the packet system headers are not modified. This is necessary for transmitting up-MEP packets after they are recycled.
   a. Create a new trap ID using bcm_rx_trap_type_create with the trap type bcmRxTrapUserDefine.
   b. Set the new trap to the OAMP destination (see above) using bcm_rx_trap_set.
   c. Create a new context for stage bcmFieldStageIngressPMF1 using bcm_field_context_create.
   d. Set context system header profile to no editing using bcm_field_context_param_set with the fields:
      - param_type=bcmFieldContextParamTypeSystemHeaderProfile
      - param_arg=0
      - param_val=bcmFieldSystemHeaderProfileNone
   e. Use the same context in bcmFieldStageIngressPMF2 in bcm_field_context_param_set with the fields
      - param_type=bcmFieldContextParamTypeSystemHeaderStrip
      - param_arg=0
      - param_val=BCM_FIELD_PACKET_STRIP(bcmFieldPacketRemoveLayerOffset0, 0);
   f. Add a trap code preselector using bcm_field_presel_set with the following fields:
      - presel_entry_id.presel_id=3
      - presel_entry_id.stage=bcmFieldStageIngressPMF1
      - presel_data.entry_valid=1

- presel_data.context_id=<context ID resulting from bcm_field_context_create>
- presel_data.nof_qualifiers=1
- presel_data.qual_data[0].qual_type= bcmFieldQualifyRxTrapCode
- presel_data.qual_data[0].qual_value=<trap code resulting from bcm_rx_trap_type_create>
- presel_data.qual_data[0].qual_mask=0x1FF

2. Create an egress trap that stamps the packet for the ingress trap.
   a. Create a new trap ID using bcm_rx_trap_type_create with the trap type bcmRxTrapEgTxOamUpMEPDest2
   b. Set the new trap using bcm_rx_trap_set with the following fields:
      - cpu_trap_gport is a gport trap (use BCM_GPORT_TRAP_SET) with the ingress trap code (see above) trap strength 15 and snoop strength 0.
      - stamped_trap_code=Standard Ethernet OAM trap code (obtained from bcm_rx_trap_type_get() with bcmRxTrapOamEthAccelerated)

On the client unit, create or configure an bcmOAMProfileEgressAcceleratedEndpoint profile.

1. Use bcm_oam_profile_action_set().
2. Assign the trap created previously to the multicast CCMs.

On the client, get the PP port for an ingress port used for the up-MEP:

1. Use bcm_vlan_port_create to create a VLAN port (this is usually done anyway for regular OAM endpoints).
2. Use bcm_vlan_port_find with the field vlan_port_id set to the above VLAN port. The resulting port is written to the field port.
3. Use bcm_port_get with the above result to get the PP port (a field in the mapping_info field pp_port).

On the server, create an up-MEP on this device as you would a regular accelerated up-MEP, with the following exceptions:

1. remote_gport is set by using BCM_GPORT_TRAP_SET with the client ingress trap code, trap strength 7 and snoop strength 0. When remote_gport is valid, the endpoint is treated as a server endpoint.
2. Set tx_gport to be the system port defined on the client recycle port.
3. Set gport to be the PP port (see above).
4. Set acc_profile_id to be invalid.

On the client create an endpoint on this device as you would a non-accelerated up-MEP, with the following exceptions:

Set the flag BCM_OAM_ENDPOINT2_HW_ACCELERATION_SET on flags2. When this flag is set, the endpoint is treated as a client endpoint.

- id – Must be the same ID as the server endpoint's ID
- flags – Set flag BCM_OAM_ENDPOINT_WITH_ID
- local_id – Must be the remote device unit where the server endpoint is optional
- group – Will not be used, but must be valid for verification
- level – Must be the same value as in server endpoint
- acc_profile_id – Should be the profile configured previously.

33.21.2.3 Server-Client Configuration

It is possible to create *client* endpoints on the server device itself. Packets sent from a remote device directly to the server device are processed and trapped to the OAMP as if they arrived from the client.

Server-client down-MEP configuration (all done on server device)

- Use `bcm_l2_station_add` to add the same MAC addresses as in the client device
- Create OAM ingress, egress, and accelerated ingress profiles.
- Create a VLAN port for the port you with to use
- Use `bcm_oam_lif_profile_set` to associate the OAM ingress and egress profiles to the VLAN port.
- Create a down-MEP similar to the one created on the client. Set `gport` value to be your VLAN port, `acc_profile_id` to be your accelerated ingress profile, and group must only be valid to pass API verification and is meaningless otherwise.

Server-client up-MEP configuration (all done on server device)

- Use `bcm_l2_station_add` to add the same MAC addresses as in the client device
- Create OAM ingress, egress, and accelerated egress profiles.
- Create a VLAN port for the port you with to use.
- Use `bcm_oam_lif_profile_set` to associate the OAM ingress and egress profiles to the VLAN port.
- Create an up-MEP similar to the one created on the client. Set `gport` value to be your VLAN port, `acc_profile_id` to be your accelerated egress profile, and group must only be valid to pass API verification and is meaningless otherwise.

33.21.3 Shell Commands

None

33.21.4 Application Reference

For a CINT example, see `oamp_server_example()` in `cint_dnx_oam_server.c`.

33.22 OAMP Performance Monitoring

OAMP supports functions for performance monitoring that allow the measurement of frame loss and frame delay between a pair of OAM endpoints.

33.22.1 Loss Measurement

Loss Measurement (LM) collects counter values applicable for ingress and egress service frames where the counters maintain a count of transmitted and received frames between a pair of MEPs.

Every local MEP has two counters:

- TxFCl – Number of frames transmitted towards the peer MEP.
- RxFCl – Number of frames received from the peer MEP.

A MEP that participates in an LM session holds an LM database entry, capturing the state of the LM session comprised of four counters which are updated on LMR reception:

- MyTx – Number of packets transmitted to the peer MEP.
- MyRx – Number of packets received from the peer MEP.

- PeerTx – Number of packets that the peer MEP transmitted to the MEP.
- PeerRx – Number of packets that the peer MEP received from the MEP.

Loss Measurement can be performed in two ways, the following sections describe.

33.22.1.1 Single-Ended LM

Y.1731 LMM/LMR packets are exchanged between the MEPs with a configured rate. Each MEP transmits LMM frames with TxCFf field containing the value of TxFCl counter at the time of the LMM transmission.

When a valid LMM PDU is received by the MEP, LMR frame is generated and transmitted to the peer MEP. It contains the following information:

- TxFCf – Copy of TxFCf of the last received LMM packet.
- RxFCf – Value of RxFCl in the transmitting MEP at the time of the last LMM received.
- TxFCb – Value of TxFCl in the transmitting MEP at the time of the LMR transmission.
- Additionally, the receiving MEP has value of RxFCl stamped within the FTMH ASE OAM (OAM-TS) extension header.

Upon receiving LMR PDU, near-end and far-end loss calculation is performed:

- Frame-Loss(far-end) = delta(TxFCf) – delta(RxFCf) = delta(MyTx) – delta(PeerRx)
- Frame-Loss(near-end) = delta(TxFCb) – delta(RxFCl) = delta(PeerTx) – delta(MyRx)

Optionally, the MEP can hold an extended LM statistics entry containing:

- LastLMFar – Number of lost transmitted packets = delta(TxFCf) – delta(RxFCf).
- LastLMNear – Number of lost received packets = delta(TxFCb) – delta(RxFCl).
- AccLmFar – Accumulated number of data transmitted = AccLmFar + LastLmFar.
- AccLmNear – Accumulated number of data received = AccLmNear + LastLmNear.
- MaxLmFar – Maximal lost transmitted packets in a LM sampling = MAX(MaxLmFar, LastLmFar).
- MaxLmNear – Maximal lost received packets in a LM sampling = MAX(MaxLmNear, LastLmNear).

33.22.1.2 Dual-Ended LM

In Proactive Loss Measurement, LM counters are piggybacked within the CCM PDU, facilitating Loss Measurement. The CCM PDU carries the following information:

- TxFCf – Value of TxFCl in the transmitting MEP at the time of the CCM transmission.
- RxFCb – Value of RxFCl in the transmitting MEP at the time of the last CCM received.
- TxFCb – Copy of TxFCf of the last CCM received.
- Additionally, the receiving MEP has value of RxFCl stamped within the FTMH ASE OAM(OAM-TS) extension header.

Upon receiving CCM PDU, near-end and far-end loss calculation is performed:

- Frame-Loss(far-end) = delta(TxFCb) – delta(RxFCb) = delta(MyTx) – delta(PeerRx)
- Frame-Loss(near-end) = delta(TxFCf) – delta(RxFCl) = delta(PeerTx) – delta(MyRx)

Optionally, the MEP can hold an extended LM statistics entry containing:

- LastLMFar – Number of lost transmitted packets = delta(TxFCb) – delta(RxFCb).
- LastLMNear – Number of lost received packets = delta(TxFCf) – delta(RxFCl).
- AccLmFar – Accumulated number of data transmitted = AccLmFar + LastLmFar.
- AccLmNear – Accumulated number of data received = AccLmNear + LastLmNear.
- MaxLmFar – Maximal lost transmitted packets in a LM sampling = MAX(MaxLmFar, LastLmFar).
- MaxLmNear – Maximal lost received packets in a LM sampling = MAX(MaxLmNear, LastLmNear).

The following limitations exist:

- `bcm_oam_loss_add()`, `bcm_oam_loss_get()`, and `bcm_oam_loss_delete()` are not supported for non-accelerated and short endpoints.
- Update from one loss measurement type to another is not allowed.
- Down-MEP injection of double-tagged piggy-back CCMs is not supported. `BCM_OAM_ENDPOINT_FLAGS2_EGRESS_INJECTION_DOWN` flag must be set on endpoint creation instead.
- LM is not supported for OAMP server.
- When dual-ended (CCM-based) LM is configured on a MEP, the MEP cannot reply to LMM packets with LMRs.
- Dual ended LM is only supported on point-to-point MEs, meaning one endpoint may only collect statistics to one remote endpoint.

33.22.2 Additional Notes for Delay Measurement Timestamps

NOTE: This section is relevant only for the BCM88690 device.

To use DM features, add the following SOC properties, the user must configure the recycle port by `bcm_oam_control_indexed_set()` with type `bcmOamControlUpMepDmRecyclePort`, the index is `core_id` and arg is recycle port.

Define two new recycle ports as header-type-in INJECTED_2_PP, with one on each core:

```
ucode_port_<recycle-port-0>=RCY.1:core_0.<recycle-port-0>
tm_port_header_type_in_<recycle-port-0>=INJECTED_2_PP
tm_port_header_type_out_<recycle-port-0>=ETH
ucode_port_<recycle-port-1>=RCY.1:core_1.<recycle-port-1>
tm_port_header_type_in_<recycle-port-1>=INJECTED_2_PP
tm_port_header_type_out_<recycle-port-1>=ETH
```

For OAMP endpoints, these are all the required configurations.

For packets injected through external HW or the CPU, packets must be injected with the format PTCHoITMHoASEoPTCH. The following cases are the same as or similar to other cases:

- Outer PTCH – Same as the Down MEP case.
- ITMH – Similar to the Down MEP case, except that the destination should be one of the recycle ports (depending on which core the Up MEP port resides).
- ASE – Same as the Down MEP-DMM/R case. The ASE offset should be the offset, in bytes, from the end of the *outer system headers*.
- Inner PTCH – Same as the other Up MEP flows.

Offloaded MEPs, 1DM, Jumbo DM, and flexible_DA (DM) do not support this implementation and, thus, are not supported on the BCM88690.

In addition, calling `bcm_oam_delay_add()` without these SOC properties returns an error; otherwise, DMM/Rs transmitted will include an incorrect timestamp.

33.22.3 Jumbo DM Frames (TLV)

Jumbo DM frames can be supported by the configure data TLV.

Caveats or limitations include the following:

- Only 1K endpoints can be supported. The MEP entry is allocated by steps of 64 with ID.
- MEP memory allocation is self contained. Short MAID and ICC MAID support 512 entries, respectively. Support for 48B MAID is also available.
- 256 MEPs can support DMM/DMR with TLV lengths of up to 9600.
- Call `bcm_oam_delay_add()` before `bcm_oam_loss_add()` for delay sessions with TLV.
- TLV length is configurable but must be a multiple of 12.
- The data pattern is a 4-byte repeatable pattern.
- The data TLV of DMR is from the local DMM configuration.
- Response packets (that is, packets that produce DMR upon incoming DMM) are available only on endpoints where jumbo DM is configured. TLV produced on DMR is configured through the API.
- This feature is not supported on MEPs created with `BCM_OAM_ENDPOINT_FLAGS2_EGRESS_INJECTION_DOWN` and related flags
- This feature is not supported for 1DM frames.
- This feature is not supported with `BCM_OAM_ENDPOINT_FLAGS2_ADDITIONAL_GAL_SPECIAL_LABEL` endpoints.

33.22.4 LM/DM Flexible DA

Normally the LMM packet and DMM packet can be transmitted with 8 × 128 different full destination MAC addresses. LM/DM Flexible DA can break this limitation. With this feature enabled, destination MAC address can be configured per MEP for LMM packet and DMM packet transmission.

LM/DM Flexible DA is enabled by setting extra data index to non-zero value, and disabled via setting extra data to zero value when creating MEP.

- This feature is not supported on MEPs created with `BCM_OAM_ENDPOINT_FLAGS2_EGRESS_INJECTION_DOWN` and related flags.
- The feature is not supported for 1DM frames.
- Jumbo DM frames are not supported with flexible DA.

33.22.5 Generating Response Packets

Upon reception of DMM/LMM/SLM/LBM packets, OAMP may generate response packets:

- DA is copied from the incoming packet's SA.
- SA is the SA configured on the MEP (same SA as transmitted CCMs).
- The remaining Ethernet header is copied from the incoming packet.
- TLVs are only generated for preconfigured jumbo DMM endpoints.

33.22.5.1 Report Mode

LM/DM statistics can be received either through the `bcm_oam_loss/delay_get()` API or through CPU events (report). This configuration is per MEP. An event is triggered per LMR/DMR/1DM reception.

The following are three types of available reports:

- Normal – Report data consists of TxFCb – RxFCb and TxFCf – RxFCf in the case of LMR, last-delay in case of DMR, and last-delay far-end when two-way stats are enabled.
- Compact – Same as above, but only 16 bits for LM stats (as opposed to 32 bits) and 32 bits for DM stats (as opposed to 64 bits).
- Raw – In this case, the raw statistics are returned directly from the packet.

Report mode type is a global configuration and is done through `bcmOamReportModeType<Normal/Compact/Raw>` in `bcm_oam_control_set`.

Similar to LOC events, callbacks are registered with `bcm_oam_performance_event_data_t`.

33.22.6 Configuration Flow

Each accelerated OAM endpoint can be associated with an LM-entry, DM-entry or LM-statistics-entry, thus having the OAMP manage loss measurement and delay measurement.

33.22.6.1 Loss Measurement

In each pair, any of the MEPs could be an initiator (transmits LMMs), a responder (receives LMMs and replies with LMRs), or both.

Configuration for all cases includes:

1. Type of the LM to be performed should be applied using `bcm_oam_profile_action_set`, described in Section 33.4, OAM Profiles and Classification.
2. Call `bcm_oam_lif_profile_set` to configure Dual-Ended/Single-Ended non-accelerated ingress and egress profiles according to LM type.

   For example, refer to `cint_oam_action.c`.
3. Call `bcm_oam_endpoint_create` – See Section 33.15, Creating OAM Endpoints.

In case the endpoint is an initiator:

4. Configure OAMP destination for LMR packets.

   For example, refer to `oam_set_opcode_destination_to_oamp()` in `cint_oam_actions.c`.
5. Call `bcm_oam_loss_t_init` and fill the `bcm_oam_loss_t` structure:
   - `flags`
     - `BCM_OAM_LOSS_STATISTICS_EXTENDED` – Enable LM-Statistics-Entry.
     - `BCM_OAM_LOSS_SINGLE_ENDED` – When set, PDUs used for loss management are LMM/LMRs, otherwise CCMs.
     - `BCM_OAM_LOSS_REPORT_MODE` – Report LM measurement or statistics on the reports interface.
     - `BCM_OAM_LOSS_UPDATE` – Update the period or the LM-Statistics mode (insert or remove an additional entry accordingly).
     - `BCM_OAM_LOSS_WITH_ID` – The entry in which to put the LM/LM_STAT data is specified explicitly in `loss_id`.
     - `BCM_OAM_LOSS_SLM` – See Section 33.17, Synthetic Loss Measurement.

- id – Endpoint ID on which the entry is configured.
- period – TX period in milliseconds. The period should be one of:
  - BCM_OAM_ENDPOINT_CCM_PERIOD_DISABLED
  - BCM_OAM_ENDPOINT_CCM_PERIOD_3MS
  - BCM_OAM_ENDPOINT_CCM_PERIOD_10MS
  - BCM_OAM_ENDPOINT_CCM_PERIOD_100MS
  - BCM_OAM_ENDPOINT_CCM_PERIOD_1S
  - BCM_OAM_ENDPOINT_CCM_PERIOD_10S
  - BCM_OAM_ENDPOINT_CCM_PERIOD_1M
  - BCM_OAM_ENDPOINT_CCM_PERIOD_10M
- peer_da_mac_address – MAC DA in LMM injection. The other L2 fields are taken from the associated local MEP encapsulation parameters given when creating a local endpoint (VLANs, TPIDs).
- loss_id – When a loss entry is added to a MEP, user must supply the MEP-DB entry to which it would be written. The range available is 4 – (12 × 8192 – 4) where all the entries between 4 and 65532 are shared with MEPs (that is, each can either host a MEP or Loss/Delay data but not both). When a loss entry is added with the STATISTICS_EXTENDED flag, two entries will be allocated: the entry provided by the user in the loss_id field along with an additional entry at 8192 + the given loss_id. When a loss entry is added prior to a delay entry, any available slot in the MEP DB will suffice. When a loss entry is added following a delay entry, the loss_id must equal 8192 + the allocated delay entry, given as delay_id in the bcm_oam_delay _add API.

NOTE: Loss entry and endpoint entry should be in different banks.

6. Call bcm_oam_loss_add to write/update the LM entry. The OAMP will start transmitting LMM frames with the desired period and collect statistics from incoming LMRs.
7. Call bcm_oam_loss_get to read the LM-entry. The call returns the following fields, all related to information recorded upon reception of LMRs:
   - rx_farend – Last peer receive frame count recorded by last LMR/CCM (represented by the RxFCf field of last received LMR in case single-ended LM or RxFCb of last received CCM otherwise).
   - tx_farend – Last peer transmit frame count recorded by last LMR (represented by the TxFCb field of last received LMR in case single-ended LM or TxFCf of last received CCM otherwise).
   - rx_nearend – Local receive frame count recorded at the time of last received LMR/CCM.
   - tx_nearend – Last local transmit frame count recorded by last LMR (represented by the TxFCf field of last received LMR in case single-ended LM or TxFCb of last received CCM otherwise).

   If loss_add has been called with the BCM_OAM_LOSS_STATISTICS_EXTENDED flag, the following fields will be filled as well:
   - loss_nearend_max – Nearend maximal loss.
   - loss_nearend – Percentage of near end packets lost (expressed in 100th percent).
   - loss_farend – Percentage of far end packets lost (expressed in 100th percent).
   - loss_nearend_acc – Near end accumulated loss. A value of -1 if not available.
   - loss_farend_max – Far end maximal loss. A value of -1 if not available.
   - loss_farend_acc – Far end accumulated loss. A value of -1 if not available.

If the endpoint is a responder:

8. Configure OAMP destination for LMM packets.

   For an example, refer to oam_set_opcode_destination_to_oamp() in cint_oam_actions.c.

33.22.6.2 Delay Measurement

In each pair, any of the MEPs could be an initiator (transmits DMMs/1DMs), a responder (receives DMMs/1DMs and replies with DMRs in case of two-way DM) or both.

If the endpoint is an initiator:

1. Configure OAMP destination for DMR/1DM packets. For example, refer to `oam_set_opcode_destination_to_oamp()` in `cint_oam_actions.c`.
2. Call `bcm_oam_delay_t_init` and fill the `bcm_oam_delay_t` structure:
   - flags
     - `BCM_OAM_DELAY_ONE_WAY` – Indicates dual-ended DM, which supports one-way stats mode only.
     - `BCM_OAM_DELAY_ONE_WAY_STATS` – Indicates single-ended DM works in one-way mode. Use this flag only when `BCM_OAM_DELAY_ONE_WAY` is not used.
     - `BCM_OAM_DELAY_REPORT_MODE` – Report DM measurement or statistics on the reports interface.
     - `BCM_OAM_DELAY_UPDATE` – Update the period.
     - `BCM_OAM_DELAY_WITH_ID` – The entry in which to put the delay data is specified explicitly in `delay_id`.
   - `id` – Endpoint id on which the delay object is configured.
   - `peer_da_mac_address` – MAC DA in DMM injection. The other L2 fields are taken from the associated local MEP encapsulation parameters given when creating a local endpoint (VLANs, TPIDs).
   - `timestamp_format` – Should be set to bcmOAMTimestampFormatIEEE1588v1.
   - `period` – TX period in milliseconds. The period should be one of:
     - `BCM_OAM_ENDPOINT_CCM_PERIOD_DISABLED`
     - `BCM_OAM_ENDPOINT_CCM_PERIOD_3MS`
     - `BCM_OAM_ENDPOINT_CCM_PERIOD_10MS`
     - `BCM_OAM_ENDPOINT_CCM_PERIOD_100MS`
     - `BCM_OAM_ENDPOINT_CCM_PERIOD_1S`
     - `BCM_OAM_ENDPOINT_CCM_PERIOD_10S`
     - `BCM_OAM_ENDPOINT_CCM_PERIOD_1M`
     - `BCM_OAM_ENDPOINT_CCM_PERIOD_10M`
   - `delay_id` – When a delay entry is added to a MEP entry, it must supply the MEP-DB entry which it allocates. The range available is 4 – (12 × 8192 – 4) where all the entries between 0 and 65532 are shared with MEPs (i.e. each can either host a MEP or Loss/Delay data but not both). When a delay entry is added following a loss entry, the `delay_id` must equal 8192 + the allocated loss entry, given as loss_id in the `bcm_oam_loss_add` API, unless the latter was called with the flag `BCM_OAM_LOSS_STATISTICS_EXTENDED`, in which case `delay_id` must equal 16384 + the allocated loss entry.

NOTE: Delay entry and endpoint entry should be in different banks.

3. Call `bcm_oam_delay_add` to write/update the DM entry – the OAMP will start transmitting DMM frames with the desired period and collect statistics from incoming DMRs.
4. Call `bcm_oam_delay_get` to read the DM-entry. The statistics the API returns depend on how `bcm_oam_delay_add()` was called:
   - If `bcm_oam_delay_add()` was called with the flag `BCM_OAM_DELAY_ONE_WAY_STATS`, the following statistics are returned:
     - delay – Last delay one-way far end of last received DMR.
     - delay_min – Minimum last delay one-way far end since time of last read.
     - delay_max – Maximum last delay one-way far end since time of last read.
     - delay_near_end – Last delay one-way near end of last received DMR.

  - delay_min_near_end – Minimum last delay one-way near end since time of last read.
  - delay_max_near_end – Maximum last delay one-way near end since time of last read.
- Otherwise, if a single-ended DM delay represents Frame-Delay (two-way) and a dual-ended DM delay represents Frame-Delay (one-way), the following statistics are returned:
  - delay - Peer receive delay measured at the time of last received DMR.
  - delay_min – Minimum delay measurement since time of last read.
  - delay_max – Maximum delay measurement since time of last read.

5. Configure OAMP destination for DMM/1DM packets.

   For an example, refer to `oam_set_opcode_destination_to_oamp()` in `cint_oam_actions.c`.

NOTE: For DMM, use the `bcm_port_phy_timesync_config_set` API to enable RX timestamping required in the port.

33.22.6.2.1 LM/DM Flexible DA

LM/DM Flexible DA can be enabled or disabled when creating MEP.

Enable Flexible DA

Configuration for flow includes:

1. Call `bcm_oam_endpoint_create` to create MEP,
   - In structure `bcm_oam_endpoint_info_t`, field `extra_data_index` is set to non-zero value,
   - The usage of extra data is managed by user,
   - For other details about creating MEP, see Section 33.15, Creating OAM Endpoints.
2. Call `bcm_oam_loss_add` to add the LM entry, see Section 33.22.6.1, Loss Measurement for details.
3. Call `bcm_oam_delay_add` to add the DM entry, see Section 33.22.6.2, Delay Measurement for details.

Update Flexible DA on the Existing LM/DM Entry

Configuration for flow includes:

1. Call `bcm_oam_loss_delete` to delete the existing LM entry,
2. Call `bcm_oam_delay_delete` to delete the existing DM entry,
3. Call `bcm_oam_endpoint_create` to update field `extra_data_index` in structure `bcm_oam_endpoint_info_t` (from zero to nonzero or from nonzero to zero),
4. Call `bcm_oam_delay_add` to add the DM entry,
5. Call `bcm_oam_loss_add` to add the LM entry.

NOTE: `extra_data_index` cannot be updated when a LM or DM entry exists on MEP.

33.22.6.2.2 Jumbo DM Frame

Call `bcm_oam_endpoint_create()` to create a MEP. Mandatory parameters are as follows:

- `flags` – `BCM_OAM_ENDPOINT_WITH_ID`
- `id` – Must be a multiple of 64
- `endpoint_memory_type` – `bcmOamEndpointMemoryTypeSelfContained`.

Call `bcm_oam_delay_add()` to add the DM entry. Mandatory parameters are as follows:

- `period` – `BCM_OAM_ENDPOINT_CCM_PERIOD_100MS` or above support for TLVs is through the TLVs struct:
  - `tlv_type` – Must be `bcmOamTlvTypeData`
  - `tlv_length` – Must be a multiple of 12
  - `four_byte_repeatable_pattern` – Data pattern is a 4-byte repeatable pattern.

33.22.7 Shell Commands

None

33.22.8 Application Reference

For a CINT example, see `cint_oam_performance_measurement.c`.

- `dnx_oam_lm_dm_run_with_defaults` with `is_jumbo_dm==1`, demonstrate how to create a jumbo DM entry and make it work.
- `dnx_oam_jumbo_dm_run_range` demonstrates how to create 1024 MEPs and 256 Jumbo DM entries.

33.22.9 API Descriptions

33.22.9.1 Loss Measurement

API Name	Highlights
`bcm_oam_loss_add(int unit, bcm_oam_loss_t *loss_info)`	Add OAM loss entry to OAMP MEP DB.
`bcm_oam_loss_get(int unit, bcm_oam_loss_t *loss_info)`	Read OAM loss entry from OAMP MEP DB.
`bcm_oam_loss_delete(int unit, bcm_oam_loss_t *loss_info)`	Free OAM loss entry from OAMP MEP DB.

33.22.9.2 Delay Measurement

API Name	Highlights
`bcm_oam_delay _add(int unit, bcm_oam_delay_t *delay_info)`	Add OAM delay entry to OAMP MEP DB.
`bcm_oam_delay _get(int unit, bcm_oam_delay_t *delay_info)`	Read OAM delay entry from OAMP MEP DB.
`bcm_oam_delay _delete(int unit, bcm_oam_delay_t *delay_info)`	Free OAM delay entry from OAMP MEP DB.

33.23 OAM over PWE/MPLS Variations

33.23.1 MPLS-TP OAM MDL Mask

For Y.1731 OAM over MPLS-TP, the maintenence domain level (MDL) must be 7, as specified in the current standards document. However, in some customer application scenarios, the MDL is not used to identify Y.1731 OAM over MPLS-TP and should be ignored.

When the MPLS-TP OMA MDL mask is enabled, the MDL is ignored on the identification of Y.1731 OAM over MPLS-TP. Otherwise, the MDL is used as the part of the key on identification of Y.1731 OAM over MPLS-TP. By default, the MPLS-TP OMA MDL mask is disabled.

To enable or disable the MPLS-TP OMA MDL mask, use the `bcm_oam_control_set()` API.

The OAM packet formats affected by this feature are as follows:

- Y.1731oACHoGALoETH (MPLS Section OAM)
- Y.1731oACHoGALoMPLSxoETH (MPLS LSP OAM)
- Y.1731oACHoGALoPWEoMPLSxoETH (PWE OAM)

33.23.2 Ingress-Only and Egress-Only Endpoints

It is possible to define one-sided MPLS endpoints. Egress-only endpoints represent endpoints where the OAMP only transmits packets, including handling LM counting. Ingress-only endpoints are endpoints where the OAMP only receives packets, including handling RX LM counters.

Egress-only endpoints may be defined by setting only the `mpls_out_gport` field (along with all accelerated fields in `endpoint_create`) but not the `gport` field.

Ingress-only endpoints may be defined by defining the `gport` field but not the `mpls_out_gport`, `intf_id`, or `egress_label` fields.

33.23.3 Custom GAL

RFC-5586 assigns the value 13 to the GAL. It is possible to use a custom value instead of value 13 for certain endpoints:

1. Use the API `bcm_switch_control_set()` with control type `bcmSwitchOamAdditionalGalSpecialLabel`, to globally determine the additional GAL value. Use value 13 to disable additional GAL.
2. In `bcm_oam_endpoint_create()`, for accelerated endpoints of type MPLS-TP or PWE with GAL, use the flag `BCM_OAM_ENDPOINT_FLAGS2_ADDITIONAL_GAL_SPECIAL_LABEL` to injects packets with the special GAL (the one configured in Step 1).

33.23.4 Hierarchical LM by LIF

To support loss measurement for injected OAM over MPLS-TP/PWE/Section, use the following sequence.

1. Identify y1731 OAM over PWE in the egress parser by calling `bcm_switch_control_indexed_set()`. Refer to `dnx_oam_identify_y1731_oam_over_pwe_egress` in `cint_dnx_utils_oam.c`.
2. Always enable a counter in the egress LIF profile by calling `bcm_oam_profile_action_set`. Refer to `dnx_oam_egress_with_update_counter_in_egr_profile()` in `cint_dnx_utils_oam.c`.

3. For injected DMM/Rs, LMM/Rs, and piggyback CCMs (dual-ended LM) over MPLS, control whether these packets should increment LM counters by calling `bcm_oam_profile_action_set()` with a reserved egress accelerated profile, based on the OpCode. Refer to `appl_dnx_oam_hlm_egress_with_update_counter_in_egr_acc_profile()` in `appl_ref_oam_init.c`. Profile 0 is the default profile, and profile 15 is reserved for injected OAM over MPLS/PWE/Section.
4. For OAM packets not mentioned in this procedure, a counter increment indication can be controlled through the egress LIF profile with `bcm_oam_profile_action_set()`.

When an injected OAM packet also hits a LIF on a lower hierarchy on which another OAM endpoint exists, the OAM packet is treated as a data packet on the lower hierarchy. For example, if there are OAM MEPs with LM on a PWE, and LSP LIF and the injected PWE OAM packet also hits the LSP LIF, the PWE OAM packets increment the LM counters on the LSP MEP.

33.24 Signal Degrade Indication

According to the IETF *Signal Degrade Indication in Segment Routing over MPLS Network* draft (https://tools.ietf.org/html/draft-han-mpls-sdi-sr-01), it is possible to control the EI bit on CCMs for OAM over MPLS-TP, PWE, and MPLS-Section.

The EI bit is part of the flags on CCMs.

To create an endpoint that supports SD monitoring/transmission:

- flag2 `BCM_OAM_ENDPOINT_FLAGS2_SIGNAL` must be set
- An extra-MEP-DB entry must be allocated. The MEP-DB entry to be allocated is `extra_data_index`.
- To transmit CCMs with the EI bit set, `tx_signal` must be set to 1 (`bcmOamEndpointSignalError`).
- The expected value of EI on received CCMs is set on `rx_signal`.

  When a CCM arrives with an EI bit that does not match `rx_signal`, the packet is punted to the CPU with the trap code `bcmRxTrapOampFlexCrcMissErr`. This action may trigger a timeout event.

33.24.1 SOC Properties

None

33.24.2 Configuration Flow

To enable or disable the MPLS-TP OMA MDL mask feature, use the following API calling sequence:

- To enable this feature, call `bcm_oam_control_set` with `type=bcmOamControlMplsOamMdlIgnore` and `arg=1`.
- To disable this feature, call `bcm_oam_control_set` with `type=bcmOamControlMplsOamMdlIgnore` and `arg=0`.

NOTE: This feature can be enabled and disabled only if no endpoints of the following type exist:

- `bcmOAMEndpointTypeBhhSection`
- `bcmOAMEndpointTypeBHHMPLS`
- `bcmOAMEndpointTypeBHHPwe`

33.24.3 Shell Commands

None

33.24.4 Application Reference

An example for HLM in egress with PWE protection is in `oam_run_hlm_egress_with_pwe_protection` in `cint_dnx_oam_y1731_over_pwe_protection.c`.

33.25 OAM Default Endpoints

When an OAM packet is parsed (even if it does not match any existing endpoint configured in the device), it can still be processed in the *default* endpoints. Currently, only default ingress endpoints are supported, and they are a resource shared with BFD. A total of four default ingress OAM endpoints and default BFD endpoints are supported. OAM packets are classified by LIF. OAM MEPs can be defined for a LIF, but if an OAM packet arrives on an in-LIF that has no MEPs, it can still be handled by default behavior, which is defined by creating the default MEPs and their corresponding default MEP profiles.

33.25.1 SOC Properties

None

33.25.2 Configuration Flow

1. Use the `bcm_l2_station_add()` API with the flag `BCM_L2_STATION_OAM` to create a range of MACs, VLANs, and ports for the default MEP to handle.
2. Use the `bcm_vlan_port_create()` API to create a VLAN port to use with the default profile. Set the criteria field as `BCM_VLAN_PORT_MATCH_PORT_CVLAN`. The `port` and `match_vlan` fields should contain a port and a VLAN covered by the L2 station created earlier.
3. Create a non-accelerated ingress profile. A new profile can be created using `bcm_oam_profile_create` or the default profile using the `bcm_oam_profile_id_get_by_type` API.
4. Use the `bcm_port_control_set()` API to set the default profile. For arguments, use the VLAN port created in Step 2, the value `bcmPortControlOamDefaultProfile`, and a number in the range 0 to 3 (corresponding to the default endpoint that will be created).
5. Create the MEP using the `bcm_oam_endpoint_create()` API. Set the `BCM_OAM_ENDPOINT_WITH_ID` flag. For the ID field, use the values `BCM_OAM_ENDPOINT_DEFAULT_INGRESS0` to `BCM_OAM_ENDPOINT_DEFAULT_INGRESS3` and set a MEP level (value 1 to 7).
6. Create a MEP profile using the `bcm_oam_lif_profile_set()` API. For arguments, use a flag in the value range `BCM_OAM_LIF_PROFILE_FLAGS_DEFAULT_MEP_0` to `BCM_OAM_LIF_PROFILE_FLAGS_DEFAULT_MEP_3` (corresponding to the MEP ID above) from the previous step, use –1 for port, use the profile from Step 3 for the ingress profile, and use –1 for the egress profile.

33.25.3 Shell Commands

None

33.25.4 Application Reference

Two examples are available:

- `default_ep_example()` creates only a default MEP.
- `oam_run_with_defaults()` creates different types of endpoints, including a default MEP (set `default_mep_lif_profile=1` to create a corresponding MEP profile).

Both examples can be found in `cint_dnx_oam.c`.

33.26 DMAC/MDL Filter

For multicast OAM packets, the 3 lower bits of the DA must match the packet's MDL for verification. When the two values do not match, the packet may be forwarded as data.

This verification can be enabled through the OAM control type `bcmOamControlEthDmacMdlMatchCheck`.

33.26.1 SOC Properties

None

33.26.2 Configuration Flow

Enable or disable the filter with `bcm_oam_control_set()` and the `bcmOamControlEthDmacMdlMatchCheck` type.

33.26.3 Shell Commands

None

33.26.4 Application Reference

None

33.27 OAM Primary VLAN

The OAM Primary VLAN feature allows the operator to set a VLAN identifier to perform an additional packet classification on an OAM LIF. This allows the operator to deploy ETH-CFM interworking with the customer, MEPs, and MIPs through the use of the OAM LIF and one additional VLAN that immediately follows.

33.27.1 SOC Properties

None

33.27.2 Configuration Flow

The OAM Primary VLAN feature based on the above OAM accelerated endpoints configuration flow.

Once all the items above have been covered and defined, the OAM Primary VLAN endpoints may be created.

1. Enable OAM Primary VLAN on AC LIF call the API `bcm_vlan_port_create(unit, &vlan_port)`.
   – `flags2` – Set to `BCM_VLAN_PORT_FLAGS2_OAM_PRIMARY_VLAN`.
2. OAM Primary VLAN on Down MEP:
   a. Allocate OAM Primary VLAN LIF by calling API `bcm_oam_primary_vlan_create(unit, &primary_vlan_info)`.
      - `primary_vlan_info.criteria` – Set to `BCM_OAM_PRIMARY_VLAN_MATCH_PORT_VLAN` or `BCM_OAM_PRIMARY_VLAN_MATCH_PORT_VLAN_VLAN`
      - `primary_vlan_info.match_port` – Local or remote physical or logical port to match
      - `primary_vlan_info.match_vlan` – Outer VLAN ID to match
      - `primary_vlan_info.match_inner_vlan` – Inner VLAN ID to match and is always 0 for `BCM_OAM_PRIMARY_VLAN_MATCH_PORT_VLAN`
   b. Set ingress LIF profiles on OAM Primary VLAN LIF by calling to API `bcm_oam_lif_profile_set(unit, flags, gport, ingress_oam_profile, egress_oam_profile)`
      - `flags` – Set to 0
      - `gport` – Set to the OAM Primary VLAN `gport`. It is the output of API `bcm_oam_primary_vlan_create()`.
      - `ingress_oam_profile`: Ingress LIF profile
      - `egress_oam_profile`: Set to `BCM_OAM_PROFILE_INVALID`
   c. Endpoints are set on OAM Primary VLAN LIF by calling API `bcm_oam_endpoint_create(unit, &endpoint_info)`
      - `endpoint_info.gport` – Set to the OAM Primary VLAN `gport`. It is the output of API `bcm_oam_primary_vlan_create()`.
3. OAM Primary VLAN on up MEP:
   a. Allocate OAM Primary VLAN LIF by calling to API `bcm_oam_primary_vlan_create(unit, &primary_vlan_info)`
      - `primary_vlan_info.flags` – Set to `BCM_OAM_PRIMARY_VLAN_UP_FACING`
   b. Set ingress LIF profiles on OAM Primary VLAN LIF by calling API `bcm_oam_lif_profile_set(unit, flags, gport, ingress_oam_profile, egress_oam_profile)`
      - `flags` – Set to 0
      - `gport` – Set to the OAM Primary VLAN gport. It is the output of API `bcm_oam_primary_vlan_create()`.
      - `ingress_oam_profile`: Set to `BCM_OAM_PROFILE_INVALID`
      - `egress_oam_profile`: Set to egress LIF profile
   c. Endpoints are set on OAM Primary VLAN LIF by calling to API `bcm_oam_endpoint_create(unit, &endpoint_info)`
      - `endpoint_info.gport` – Set to the OAM Primary VLAN `gport`. It is the output of API `bcm_oam_primary_vlan_create()`.
4. Field Processor group for OAM Primary VLAN on UP MEP: See Chapter 11, Field Processor for general details about field processor and Application Reference for more information.

33.27.3 Shell Commands

None

33.27.4 Application Reference

An example on how to configure the OAM Primary VLAN on Up MEP and Down MEP can be found in `cint_dnx_oam_primary_vlan.c`, where `oam_primary_vlan_example(). cint_field_oam_primary_vlan.c` contains ePMF functions that are needed for OAM Primary VLAN on Up MEP.

33.27.5 API Descriptions

33.27.5.1 OAM Primary VLAN

API Name	Highlights
`bcm_oam_primary_vlan_create(int unit, bcm_oam_primary_vlan_info_t * primary_vlan_info)`	Create an OAM primary VLAN object.
`bcm_oam_primary_vlan_get(int unit, bcm_oam_primary_vlan_info_t * primary_vlan_info)`	Get an OAM primary VLAN object information with given match criteria. Only for Down MEP.
`bcm_oam_primary_vlan_destroy(int unit, bcm_oam_primary_vlan_info_t * primary_vlan_info)`	Remove an OAM primary VLAN object.
`bcm_oam_primary_vlan_traverse(int unit, bcm_oam_primary_vlan_traverse_cb trav_fn, void *user_data)`	Traverse the set of OAM primary VLAN objects and call a callback function with the data for each entry found. Only for Down MEP.

Chapter 34: BFD

34.1 Introduction

The DNX family devices perform BFD by the following entities:

- The OAM classifier within the packet processing-pipe: used for trapping BFD traffic by matching discriminators (for single-hop, micro-BFD, multi-hop packets or LSP packets) or according to LIF for MPLS-TP and PWE packets.
- The OAM Processor (OAMP) block within the device: may be used to manage BFD traffic.

The following types of BFD are supported:

- BFD over IPv4 and IPv6 single hop, including BFD Echo (RFC-5881)
- BFD over IPv4 and IPv6 multi hop (RFC-5883)
- Micro BFD (RFC-7130)
- BFD over LSP (RFC-5884)
- BFD for VCCV with PW-ACH encapsulation (without IP/UDP headers) (RFC-5885).

BFD packets may be recognized and classified in the ingress pipeline according to the packet's Your-Discriminator (or InLIF with BFD for VCCV) and trapped to the OAMP block for processing.

The OAMP supports asynchronous mode BFD, which is the transmission and reception of BFD control packets. If the Detection Time expires, the connection is regained or packets are received with different State, Diag, Flags, or Detect Multiplier fields, and the user is notified through events or protection packets.

34.2 Application Configuration Checklist

- BFD Traps
- MPLS/PWE channel type configuration
- OAMP MEP-DB management (TBD)
- OAMP punt packets
- BFD events and protection packets
- BFD endpoints
- Encapsulation-types and Encapsulation

34.3 BFD Traps

The OAM/BFD classifier uses traps to redirect packets to the OAMP/CPU/external BFD processing unit. Before creating an endpoint the following traps should be configured:

- `bcmRxTrapBfdOamDownMEP` – Default trapping for packets not processed in OAMP. Note: The trap is shared with OAM.
- `bcmRxTrapOamBfdIpv4` – For IPv4 single-hop/mult-hop/micro-BFD endpoints. The destination trap must be set to the local OAMP.
- `bcmRxTrapOamBfdIpv6` – For IPv6 single-hop/mult-hop/micro-BFD/echo endpoints. The destination trap must be set to the local OAMP.
- `bcmRxTrapOamBfdMpls` – For BFD over MPLS. When using the OAMP for BFD processing, the destination trap must be set to the local OAMP.
- `bcmRxTrapOamBfdPwe` – For BFD over PWE. The destination trap must be set to the local OAMP.

  If additional destinations are required, user-defined traps may be used.
- `bcmRxTrapEgBfdIpv6InvalidUdpChecksum` – Filters BFD IPv6 packets with incorrect UDP checksum (despite the name, it is an ingress trap).
- `bcmRxTrapOamBfdCcMplsTp` – Used for endpoints of type `bcmBFDTunnelTypeMplsTpCcCv`.

See Section 34.10, Creating BFD Endpoints to see how the traps may then be used.

When endpoints are created with `bcm_bfd_endpoint_create()`, by default, traps are created with trap strength taken from `dnx-data bfd_default_trap_strength` (default value is 7) and snoop strength taken from `dnx-data bfd_default_snoop_strength` (default value is 0).

To create an endpoint with a different trap/snoop strength, set the `remote_gport` field with the desired trap and strength.

34.3.1 SOC Properties

None

34.3.2 Configuration Flow

For the full calling sequence, see Chapter 12, Traps. `bcm_rx_trap_set()` must be called as follows:

- `trap_config.flags` must be set with the following flags:
  - `BCM_RX_TRAP_TRAP` – Set the system headers of the trapped packet as expected by the OAMP.
  - `BCM_RX_UPDATE_DEST` – Update the trapped destination of the packet.
  - `BCM_RX_TRAP_UPDATE_EGRESS_FWD_INDEX` – Required for OAMP processing.
- `trap_config.egress_forwarding_index` must be set to 7.
- When using BCM88670 (Jericho 1) system headers mode, do not use `BCM_RX_TRAP_UPDATE_EGRESS_FWD_INDEX`, `egress_forwarding_index`. Instead use `BCM_RX_TRAP_UPDATE_FORWARDING_HEADER` and set `forwarding_header` to `bcmRxTrapForwardingHeaderOamBfdPdu`.
- For `bcmRxTrapOamBfdIpv6` trap, use both `BCM_RX_TRAP_UPDATE_EGRESS_FWD_INDEX` and `BCM_RX_TRAP_UPDATE_FORWARDING_HEADER` and set both `egress_forwarding_index` and `forwarding_header` to 7 (use this configuration for both Jericho1 and Jericho2 system headers mode).
- For the `bcmRxTrapBfdIpv6InvalidUdpChecksum` trap, Only use the flags `BCM_RX_TRAP_TRAP`, `BCM_RX_UPDATE_DEST`. In addition, set the trap strength when creating the trap using the `trap_strength` field.
- For the `bcmRxTrapOamBfdCcMplsTp` trap, `egress_forwarding_index` and `forwarding_header` must be set to 5.

To set an endpoint with a trap strength other than 7, set the trap in the `remote_gport` field with the desired strength:

- Get the trap with `bcm_rx_trap_type_get()`.
  - For BFD IPv4 endpoints, use the trap `bcmRxTrapOamBfdIpv4`
  - For BFD IPv6 endpoints, use the trap `bcmRxTrapOamBfdIpv6`

Set the trap and strengths in the `remote_gport` field in the following manner:

- `BCM_GPORT_TRAP_SET(remote_gport , trap_code, trap_strength, snoop_strength)`
- Call `bcm_bfd_endpoint_create()` with all other fields set according to the Section 34.10, Creating BFD Endpoints.

34.3.3 Shell Commands

None

34.3.4 Application Reference

BFD Traps application reference

Example of application configuration of the different BFD traps.

- **Type:** Default Application Reference
- **Path:** `$SDK/src/appl/reference/dnx/appl_ref_oam_init.c`
- **Function**: `appl_dnx_bfd_trap_init()`

34.4 MPLS/PWE Channel Type Configuration

This subject is covered by Section 33.7, MPLS/PWE Channel Type Configuration.

34.5 OAMP MDB and MEP-DB Management

The endpoint's Your-Discriminator (`local_discr in bcm_bfd_endpoint_create()`) 16 lower bits determine the endpoint-ID (OAM-ID). This implies the following:

- It is possible to determine whether an endpoint is self-contained or short through the `local_discr`. If the 16 lower bits of the `local_discr` are lower than `oamp_mep_full_entry_threshold`, the endpoint will be short. Otherwise, it will be self contained.
- If the endpoint's 16 lower bits of the `local_discr` are greater than the threshold, the `local_discr` must be a multiple of 4.
- Only IDs 0 to 65,535 may be used.
- For BCM88670 compatibility mode, only self-contained endpoints are supported. This means that `oamp_mep_full_entry_threshold` must be set to 0.

34.6 BFD IPv6 in MEP DB

BFD over IPv6, including BFD over LSP, is implemented in the OAMP. The pointer to the first entry is provided in the `endpoint_create()` API through `ipv6_extra_data_index`. The subsequent entry is allocated at the same offset in the next bank. For example, if `ipv6_extra_data_index` is set to 1000 in `endpoint_create()`, entries (1000 + i × 4096) for i=0,1 are allocated in the MEP-DB.

34.7 OAMP Punt Packet

This feature is described in Section 33.11, OAMP Punt Packet. Similar to OAM packets, BFD packets are also subjected to validity checks by the OAMP and failed packets are punted. Below is the list of BFD-relevant OAMP punt traps:

- `bcmRxTrapOampMyDiscErr` – Packet arrived at the OAMP with a My-Discriminator field miss.
- `bcmRxTrapOampSrcIpErr` – Packet arrived at the OAMP with a source IPv4 address miss.
- `bcmRxTrapOampYourDiscErr` – Packet arrived at the OAMP with a Your-Discriminator field miss. If the MEP-ID does not equal the endpoint's My-Discriminator (see Section 34.12, Endpoint Types and Encapsulation), then this verification is disabled.
- For BFD over IP, LSP endpoints the classification is based on the Your-Discriminator, so the verification is redundant.
- `bcmRxTrapOampSrcPortErr`– Packet arrived at the OAMP with a UDP source port miss.
- `bcmRxTrapOampRmepStateChange` – Packet arrived at the OAMP that does not match the current RMEP state. Applied only on endpoints where `sampling_ratio` is non-zero.
- `bcmRxTrapOampFlexCrcMissErr` – Packet arrived at the OAMP with a source IPv6 address miss (OAMP verifies source-IP using CRC validation). This trap also applies to Your-Discriminator validation for BFD over PWE.

34.8 BFD Events and Protection Packets

Protection packets are described in Section 33.13, OAM Events and Protection Packets. BFD events share the same event types as OAM event (the types relevant to BFD, that is) and BFD events can be lumped together with OAM events in a protection packet.

Events may be used by registering callbacks.

34.8.1 SOC Properties

None

34.8.2 Configuration Flow

To register a callback triggered define the callback structure:

```
typedef int (*bcm_bfd_event_cb)(
    int unit,
    uint32 flags,
    bcm_bfd_event_types_t events,
    bcm_oam_endpoint_t endpoint,
    void *user_data);
```

Parameter information is as follows:

- `events` – A bitmap of one or more of:
  - `bcmBFDEventEndpointTimeout`– BFD timeout
  - `bcmBFDEventEndpointTimein`– Loss of Continuity reset
  - `bcmBFDEventStateChange` – BFD State, Diagnostics Flags or Detect Multiplier changes. Note that only the Flags P (Poll), F (Final), C (Control Plane Independent) and D (Demand) are supported, excluding a combination of all 4. An unsupported Flag combination may also trigger a `bcmBFDEventStateChange` event, but only if the combination of flags from the previous RX BFD frame was one of the supported options.
- `endpoint` – The Remote ID (as allocated by bcm_oam_endpoint_create)
- `user_data` – Associated data with the event, supplied by the user, to be passed back to the callback.

Upon reception of a `bcmBFDEventStateChange` the Diagnostics, Flags, State and Remote Detect Multiplier may be gotten through `bcm_bfd_endpoint_get()`. These values will be returned through the `remote_diag`, `remote_flags`, `remote_state` and `remote_detect_mult` fields, respectively. Note that for unsupported flags the value of the `remote_flags` field is -1.

For events to be triggered, events must be registered using the API `bcm_oam_event_register()`:

```
int bcm_bfd_event_register(int unit,
    bcm_bfd_event_types_t event_types,
    bcm_bfd_event_cb cb, void *user_data);
```

`event_types` may be set using the macro `BCM_BFD_EVENT_TYPE_SET`, thus allowing for one callback to be configured for different events.

Events may be unregistered using the API `bcm_bfd_event_unregister`().

34.8.3 Shell Commands

None

34.8.4 Application Reference

- `cint_sand_bfd.c`
  `register_events(int unit)`

34.9 My-BFD-DIP

Incoming multi-hop BFD PDUs must have a destination-IP address matching the one configured through `src_ip_addr` or `src_ip6_addr` in `bcm_bfd_endpoint_create()`. Because there are only 16 allowed values, it is possible to avoid this verification through My-BFD-DIP.

The user may set a special `BFD_MY_DIP_DESTINATION` value (using the API `bcm_switch_control_set` with `type=bcmSwitchBfdMyDipDestination`). The value set is limited to the legal FEC values 0–0xBFFFF, and this value must not be used as an FEC elsewhere. This is done once. Then, for each endpoint created with the flag `BCM_BFD_ENDPOINT_FLAGS2_USE_MY_DIP_DESTINATION` set, an L3 route should be added from the intended destination IP (IPv4 or IPv6) to the BFD FEC, using the API `bcm_l3_route_add`. Because this API supports masking, multiple IPs can be routed to the FEC with one call.

Use the `bcm_l3_route_add()` API and set `BCM_L3_FLAGS2_SCALE_ROUTE` in `info.l3a_flags2`.

This also applies when IP forwarding is done using an external KBP. The my-BFD identification entries will reside in KAPS.

Limitations:

- An example of a supported predefined MDB profile that can support this feature is L2-XL, L3-XL.
- PBR (VRF update) cannot be used alongside my-BFD identification

34.9.1 SOC Properties

None

34.9.2 Configuration Flow

1. Set a *special* My-DIP-destination value through `bcm_switch_control_set` and the enum `bcmSwitchBfdMyDipDestination`**.** The `arg` parameter should be set to the special FEC.
2. Per each accepted source IP address, use `bcm_l3_route_add` to set the packet's destination to the special FEC set in the previous step. To use private routing, use the flag `BCM_L3_FLAGS2_SCALE_ROUTE` and set the `vrf` field.
3. Create an endpoint with the flag2 `BCM_BFD_ENDPOINT_FLAGS2_USE_MY_DIP_DESTINATION` and do not set the `dst_ip_addr` field unless the endpoint is accelerated.

34.9.3 Shell Commands

None

34.9.4 Application Reference

For a usage example, refer to `cint_sand_bfd.c` under `use_my_bfd_dip_destination`.

34.10 Creating BFD Endpoints

Three types of BFD endpoints exist:

1. Standard non accelerated endpoints: BFD PDUs are trapped according to either discriminator or gport. The classifier will identify the discriminator or gport and trap the PDU accordingly. No OAMP transmissions.
2. Standard accelerated endpoints: BFD packets are transmitted periodically from the OAMP and incoming BFD packets are monitored for loss of continuity (LoC) and state change. The different options available for accelerated BFD endpoints are described below.
3. Non accelerated endpoints; incoming BFD packets are classified by ACC MEP-DB. This allows the `oam-id` to be set on the FHEI of trapped packets. Such endpoints can be configured by setting the flag `BCM_BFD_ENDPOINT_IN_HW`, `remote_gport` to the trap destination and `tx_gport` to NULL.

For each local accelerated BFD endpoint there is only one remote endpoint, so they are created together using one API. The remote BFD endpoint has similar functionalities to remote OAM endpoints:

- Monitoring LoC status by maintaining a timer from last received BFD packet. The main difference here is how LoC is determined (explicit LoC time vs. detect_mult, and so on) When an LoC occurs or is cleared, an event may be generated.
- Monitoring the BFD state, and detecting changes. If the BFD state is changed, an event may be generated (see traps above.)

34.10.1 OAMP Transmission Periods

Accelerated BFD EPs can transmit packets in up to eight different rates.

Updating the TX rate is normally done per endpoint, call `bcm_bfd_endpoint_create()` with the `_UPDATE` flag set and the `bfd_period` field set to the desired TX rate. It is also possible to update the TX rate of many endpoints in one fell swoop. This is done by associating a `bfd_period_cluster` to an endpoint. When an endpoint's TX rate is updated, all other existing endpoint with the same `bfd_period_cluster` will be updated with the same TX rate.

Limitations to using `bfd_period_cluster`:

- Only values 1 to 8 may be used.
- Each cluster occupies a transmission rate profile, so non-clustered EPs have fewer possible values left (it's possible for different clusters to have the same rate, but they will not share a transmission rate profile.)
- When adding a new EP to a cluster, that EP must have the same `bfd_period` value as the other endpoints in the cluster. A different value will cause an error. Only when updating an endpoint, `bfd_period` is allowed a different value (which will then affect all EPs in the cluster.)
- Other values that must be the same for creating a new EP and associating it with an existing cluster:
  - Discriminator MSB. Bits 30-16 must be the same for all EPs in a device, but bit 31 can have the values 0 or 1. In a cluster, having one EP with bit 31 set to 0, and another EP with bit 31 set to 1 is forbidden
- In `flags2`, the flags `BCM_BFD_ENDPOINT_FLAGS2_TX_STATISTICS` and `BCM_BFD_ENDPOINT_FLAGS2_RX_STATISTICS` must be consistent. For example, if one EP has `BCM_BFD_ENDPOINT_FLAGS2_TX_STATISTICS` set and `BCM_BFD_ENDPOINT_FLAGS2_RX_STATISTICS` cleared, they must set and cleared respectively in all of that cluster's endpoints.

34.10.2 Remote Event Handling

Upon LoC or state change (Diag/State/Flags/Detect-Multiplier in BFD PDU) event, it is possible to determine how the OAMP generates an event.

In the RMEP, this is done through the following flags:

1. Upon LOC or LOC-clear event, the RMEP can perform the following actions (only one may be chosen), controlled through flags:
   a. `BCM_BFD_ENDPOINT_REMOTE_UPDATE_STATE_DISABLE` – Do not update fault state, but generate an event. This means that events will be generated repeatedly until the CPU updates faults in the RMEP flag through `bcm_bfd_endpoint_create()`.
   b. `BCM_BFD_ENDPOINT_REMOTE_EVENT_DISABLE` – Update fault field, but do not generate an event (note that this is in flags, not flags2).
   c. `BCM_BFD_ENDPOINT_RDI_AUTO_UPDATE` – Update fault fields unconditionally and generate an event. Events may be lost in this case.
   d. If no flag is specified – Update fault state conditionally (If event FIFO is not full), and generate event.
2. Upon Diag/State/Flags/Detect-Multiplier change in BFD PDU from last recorded state in the RMEP, the RMEP can perform the following actions (only one may be chosen), controlled through flags:
   a. `BCM_BFD_ENDPOINT_REMOTE_UPDATE_STATE_DISABLE`

      Do not update Diag/State/Flags/Detect-Multiplier value, but generate an event. This means that events will be generated repeatedly until the CPU updates the respective fields in the MEP through `bcm_oam_endpoint_create()`.

b. `BCM_BFD_ENDPOINT_RX_REMOTE_EVENT_DISABLE`

Update Diag/State/Flags/Detect-Multiplier value, but do not generate an event.

c. `BCM_BFD_ENDPOINT_RX_RDI_AUTO_UPDATE`

Update Diag/State/Flags/Detect-Multiplier value unconditionally and generate an event.

d. If no flag is specified – Update state/RDI conditionally (If event FIFO is not full), and generate an event.

In items 1 and 2, The difference between option (c) and (d) is that if the OAMP event FIFO is full, in option (c), the RMEP state will be updated but an event will not be generated. In option (d), the RMEP state will not be updated, until the event FIFO is not full, at which point an event will be generated and the RMEP state will be updated. Options (b) or (c) should be used when relying on protection packet for information on incoming events. Option (d) should be used when relying on interrupts. Option (a) may be used for debugging.

All five items, together with the `sampling_ratio` determine a `punt_profile`. There are 16 punt profiles available, so 16 combinations of the six fields are possible. Punt profiles are shared with OAM.

34.10.3 SOC Properties

None

34.10.4 Configuration Flow

Call `bcm_bfd_endpoint_info_t_init()` and fill a `bcm_bfd_endpoint_info_t` struct according to the type of endpoint. Then call `bcm_bfd_endpoint_create()`. Parameters are described below.

RX fields (Non-accelerated and Accelerated)

- `id` – Endpoint identifier, output field.
- `type` – type of BFD endpoint. Supported types:
  - `bcmBFDTunnelTypeUdp` – BFD over IPv4/IPv6, including single hop, multi hop and micro BFD (see flags)
  - `bcmBFDTunnelTypeMpls` – BFD over MPLS LSP
  - `bcmBFDTunnelTypePweControlWord` – BFD over PW with CW
  - `bcmBFDTunnelTypePweRouterAlert` – BFD over PW with Router Alert
  - `bcmBFDTunnelTypePweTtl` – BFD over PW with TTL=1
  - `bcmBFDTunnelTypeMplsTpCc` – BFD for MPLS-TP proactive CC. For this encapsulation type, a BFD over PWE entry will be created, and not BFD over MPLS.
  - `bcmBFDTunnelTypePweGal` – BFD over PW with GAL, GACH
  - `bcmBFDTunnelTypeMplsTpCcCv` – BFD VCCV Type 1 with IP/UDP encapsulation
  - `bcmBFDTunnelTypeMplsPhp` – BFD over MPLS LSP and PHP
- `gport` – gport associated with BFD endpoint. Required for endpoints of type MPLS and PWE.
- `remote_gport` – trapping destination: OAMP or CPU, local or remote
- `src_ip_addr` – Incoming packet's destination IPv4 address (endpoint's source IP address). Required only for multi-hop endpoints. Incoming multi-hop BFD PDUs must have a source IP address matching that set through this parameter, unless `BCM_BFD_ENDPOINT_FLAGS2_USE_MY_DIP_DESTINATION` flag is set. For accelerated endpoints, this is also the source IP address in built IPv4 headers. See Section 34.9, My-BFD-DIP.
- `src_ip6_addr` – Incoming packet's destination IPv6 address (endpoint's source IP address). Required only for multi-hop endpoints. This parameter is analogous to `dst_ip_addr`.
- `local_discr` – Local discriminator. Required for UDP endpoints.

- flags
  - BCM_BFD_ENDPOINT_MICRO_BFD
  - BCM_BFD_ENDPOINT_IPV6
  - BCM_BFD_ENDPOINT_MULTIHOP
- flags2 – Additional flags
  - BCM_BFD_ENDPOINT_FLAGS2_USE_MY_DIP_DESTINATION. See Section 34.9, My-BFD-DIP.
- profile_id – Should be used on endpoints that are classified by LIF, not Your-Discriminator. This prevents trapping of OAM packets with Level 0 on the same LIF (gport) as the BFD endpoint. How to use:
  - Create a new action profile by calling API bcm_oam_profile_create(), then a new OAM profile ID is allocated.
  - Create the BFD endpoint by calling API bcm_bfd_endpoint_create() with setting profile_id above.

OAMP fields (Accelerated Only)

In addition to the non accelerated fields, the following fields are supported:

- flags
  - BCM_BFD_ENDPOINT_WITH_ID – The ID is an input field. If the full entry threshold is 0 and the device is in JR2 mode, creating a BFD endpoint with an ID is supported. In that case, local_discr is decoupled from the endpoint ID, and the low 16 bits of local_discr can be any value.
  - BCM_BFD_ENDPOINT_REMOTE_WITH_ID – See below.
  - BCM_BFD_ENDPOINT_REMOTE_EVENT_DISABLE – To be discussed in future versions.
  - BCM_BFD_ENDPOINT_RX_REMOTE_EVENT_DISABLE – To be discussed in future versions.
  - BCM_BFD_ENDPOINT_REMOTE_UPDATE_STATE_DISABLE – To be discussed in future versions.
  - BCM_BFD_ECHO – To be discussed in future versions.
  - BCM_BFD_ENDPOINT_EXPLICIT_DETECTION_TIME – See below.
  - BCM_BFD_ENDPOINT_FLAGS2_SINGLE_HOP_WITH_RANDOM_DIP – For BFD over LSP encapsulation when the user wants to build the LSP label through the egress. In this case, the bcmBFDTunnelTypeUdp type should be used.
- remote_id – Endpoint identifier. Remote endpoint identifier. If the flag BCM_BFD_ENDPOINT_REMOTE_WITH_ID is set, this field is an input; otherwise it is an output field. In the BFD client-server model, for BFD client endpoint's remote_id may also be used as the server's unit.
- tx_gport – Tx gport associated with this endpoint. See Section 34.12.1, ITMH.
- egress_if – Egress interface. See Section 34.12.1, ITMH.
- int_pri – Egress queuing for outgoing BFD to remote. See Section 34.12.1, ITMH.
- bfd_period – For local endpoints, this is the BFD transmission period in milliseconds.
- bfd_period_cluster – If set to 0, the field will be ignored. However, if it is set to a value in the range 1 to 8, then the endpoint's period will be clustered with other endpoints using the same bfd_period_cluster, allowing one endpoint to update the TX period of all endpoints in the cluster at once. See Section 34.10.1, OAMP Transmission Periods.
- local_state – Local session state. See Section 34.12.3.4, BFD PDU.
- local_diag – Local diagnostic code. See Section 34.12.3.4, BFD PDU.
- local_flags – Flags combination on outgoing frames. See Section 34.12.3.4, BFD PDU.
- local_min_tx – Desired local min tx interval. See Section 34.12.3.4, BFD PDU.
- local_min_rx – Required local rx interval. See Section 34.12.3.4, BFD PDU.
  If BCM_BFD_ENDPOINT_EXPLICIT_DETECTION_TIME flag is not set, BFD Detection Time will be local_min_rx × remote_detect_mult (in microseconds).
- local_min_echo – Local minimum echo interval. See Section 34.12.3.4, BFD PDU.
- local_detect_mult – Local detection interval multiplier. See Section 34.12.3.4, BFD PDU.
- remote_flags – Remote flags. Set the expected incoming Flags. See Section 34.8, BFD Events and Protection Packets.

- `remote_state` – Remote session state. Set the expected incoming State. See Section 34.8, BFD Events and Protection Packets.
- `remote_discr` – Remote discriminator. See Section 34.12.3.4, BFD PDU.
- `remote_diag` – Remote diagnostic code. Set the expected incoming Diag. See Section 34.8, BFD Events and Protection Packets.
- `remote_detect_mult` – Remote detection interval multiplier. Set the expected incoming Detect Multiplier. See Section 34.8, BFD Events and Protection Packets.
- `sampling_ratio` – Rate of punt packets transmitted per state change event.
  - 0 – No packets sampled to the CPU.
  - 1 to 8 – Count of packets (with events that need to arrive before one is sampled to the CPU.)
    - 1 – Punt every state change event.
    - 8 – Punt one of every eight events.
- `loc_clear_threshold` – Number of packets required to reset the Loss-of-Continuity status per endpoint.
- `bfd_detection_time` – BFD Detection Time, in microseconds, used if `BCM_BFD_ENDPOINT_EXPLICIT_DETECTION_TIME` is set.

The following fields apply to IP endpoints:

- `dst_ip_addr` – Source IPv4 address. See Section 34.12.3.1, IPv4 Header.
- `dst_ip6_addr` – Source IPv6 address. Refer to Section 33.15.2, Endpoint Types and Encapsulation.
- `ip_tos` – IPv4 ToS / IPv6 Traffic Class. See Section 34.12.3.1, IPv4 Header.
- `ip_ttl` – IPv4 TTL / IPv6 Hop Count. See Section 34.12.3.1, IPv4 Header.
- `udp_src_port` – UDP source port for IPv4, IPv6. See Section 34.12.3.3, UDP Header.
- `ip_subnet_length` – The subnet length for incoming packet validity check. Value 0 indicates no check is performed, positive values indicate the amount of MSBs to be compared.
- `ipv6_extra_data_index` – Pointer to the extra data dedicated for the IPv6 info. For self contained accelerated BFD endpoint over IPv6, `ipv6_extra_data_index` must be set to a non-zero value greater than or equal to the OAMP MEP full entry threshold and multiple of four.

  The OAMP may optionally check the MEP's discriminator through the flexible CRC mechanism for BFD over PWE type. `ipv6_extra_data_index` should also be provided with extra data entry. In this case, one additional MEP-DB entry is used. Incorrect packets are punted with `bcmRxTrapOampFlexCrcMissErr`.
- Faults:

  `BCM_BFD_ENDPOINT_REMOTE_LOC` – When getting an endpoint, this flag indicates that the endpoint is in loss of continuity state. When creating an endpoint, if the flag is clear, an LOC event will not be triggered until the first valid packet has been received. When creating an endpoint with this flag set, the LOC event will be triggered after the configured time has elapsed (`bfd_detection_time` if the flag `BCM_BFD_ENDPOINT_EXPLICIT_DETECTION_TIME` is set, or `local_min_rx` × `remote_detect_mult` otherwise). When updating an endpoint not in the LOC state and setting this flag, an LOC event will also be triggered after the configured time elapses.

The following fields apply to MPLS/PWE endpoints:

- `label` – PWE label
- `egress_label` – The MPLS outgoing label information. See Section 34.12, Endpoint Types and Encapsulation, under LSP label and PWE label.

34.10.5 Shell Commands

`OAM BFDEndPoint [ID=<...>]`

- **Description** – Presents configured OAM BFD endpoints and their properties. When ID is supplied, a specific endpoint is presented. When used without a specific ID, all configured BFD endpoints are presented.
- **Arguments** – ID (int32:-1). The ID of the requested BFD Endpoint.
- **Examples:**
  - `OAM BFDEndPoint`
  - `OAM BFDEndPoint ID=3`

It is also possible to print the software state associated with `bfd_period_cluster` by using the following command:

```
swstate algo_bfd bfd_tx_period_cluster dump
```

34.10.6 Application Reference

TBD

34.11 OAMP Statistics

The OAMP statistics count BFD packets received and transmitted by the OAMP.

In the BCM88690, OAMP has direct access to CRPS. This enables OAMP to provide statistics functionality per endpoint. The user may configure the OAMP to count the following items:

- Packets to and from specific MEPs
- RX packet, TX packets, or both

For each packet received or transmitted by the OAMP, a counter pointer is calculated. Counting of a specific packet type may be enabled depending on its direction (Tx/Rx) and OpCode. Per OpCode counting may be configured.

NOTE: The OpCode feature is disabled for BFD.

The `counter_id` is calculated by the following formula, and the user must configure the range according to the formula. The meaning of each item is as follows:

- `tx_counter_id` = MEP-ID << Mep-Profile.Mep-Id-Shift + Opcode-For-Count-Index
- `rx_counter_id` = 1 << CFG_RX_SHIFT + MEP-ID << Mep-Profile.Mep-Id-Shift + Opcode-For-Count-Index

NOTE: For BFD, the Mep-Profile.Mep-Id-Shift and Opcode-For-Count-Index are equal to zero.

34.11.1 Counting Packets with Specific OpCodes

OAMP can count different OpCodes on separate counters. Even for BFD, specific BFD OpCode counting must be allowed.

The counting per OpCode may be configured globally through the `bcm_oam_control_index_set()` API:

- Set the type to be `bcmOamControlOampStatsTxOpcode` or `bcmOamControlOampStatsRxOpcode`
- Use `value` to determine whenever to count or not. A value of 0 disables the counting, and a value of 1 enables the counting.
- Set the `index` with the actual OpCode

NOTE: Each OpCode must be enabled or disabled per direction.

The following table shows how to calculate the counter_id.

Actual Packet OpCode	Opcode-For-Count-Index
CCM/BFD	0

Example: cint: cint_oam_oamp_statistics_main() in cint_oam_statistics.c

34.11.2 Counter Resource Management

Counter resources are managed using counter databases mapped to counter engines. Counters are accessed using stat commands, each one having a command_id.

OAM counter resources are managed by OAM-dedicated counter databases. OAM-dedicated counter databases, link OAM counter_if to a specific stat_command_id. OAMP counting is performed using command_id 0. Therefore, OAM databases should be defined using command_id.

Example: cint: cint_oam_oamp_statistics_main() in cint_oam_statistics.c

34.11.3 Configure PREFIX and SHIFT Values per Direction

The value for CFG_RX_SHIFT may be configured globally through the bcm_oam_control_set() API.

- Set the type to be bcmOamControlOampStatsShift
- Use value to set the SHIFT value.

Example: cint : cint_oam_oamp_statistics_main() in cint_oam_statistics.c.

34.11.4 Endpoint Create with OAMP Statistics

To enable the statistics feature per endpoint, create an accelerated MEP and one of the statistics flags2:

- BCM_BFD_ENDPOINT_FLAGS2_TX_STATISTICS
- BCM_BFD_ENDPOINT_FLAGS2_RX_STATISTICS

Create the endpoint with the WITH_ID flag. The Endpoint ID should be in a range based on allocated counter IDs.

34.11.5 SOC Properties

None

34.11.6 Configuration Flow

The configuration flow is as follows:

1. BFD statistics counting should be enabled globally per device. This should be done with the bcm_oam_control_indexed_set API:
   a. Set type to BFD_OPCODE.
   b. Set value to 1 (TRUE).
2. Allocate counter resources by using counter databases which are mapped to counter engines. This could be done with cint_oam_oamp_statistics_main().
   a. Set the mep_id to the first MEP that will use the statistics.

b. Set the `nof_statistics_meps`, which are the number of MEPs that will use the statistics.
c. Set the `tx_rx` to enable TX or RX counting direction.

3. Create an endpoint with the statistic flags set. This should be done with the `bcm_bfd_endpoint_create()` API.
   a. Set the `opcode_flags` to `BCM_OAM_ENDPOINT_FLAGS2_TX_STATISTICS` or `BCM_OAM_ENDPOINT_FLAGS2_RX_STATISTICS`.

34.11.7 Shell Commands

None

34.11.8 Application Reference

For examples, refer to the following functions:

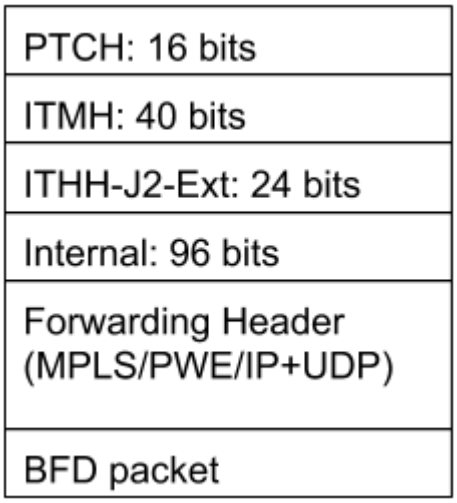
- `cint_oam_oamp_statistics_main()` in `cint_oam_statistics.c`
- `bfd_ipv4_run_with_defaults()` in `cint_sand_bfd.c`
- `bfd_ipv6_run_with_defaults()` in `cint_sand_bfd_ipv6.c`
- `bfd_pwe_run_with_defaults()` in `cint_sand_bfd.c`

34.12 Endpoint Types and Encapsulation

For accelerated endpoints the OAMP transmits BFD packets according to input from `bcm_bfd_endpoint_create()`. The following section describes how packets are encapsulated for each endpoint type, as well as the limitations on each field.

The OAMP supports six types of BFD endpoints: single-hop IPv4, multi-hop IPv4, single-hop IPv6, multi-hop IPv6, MPLS, and PWE. This is the structure of BCM88690 BFD packets originating in the OAMP.

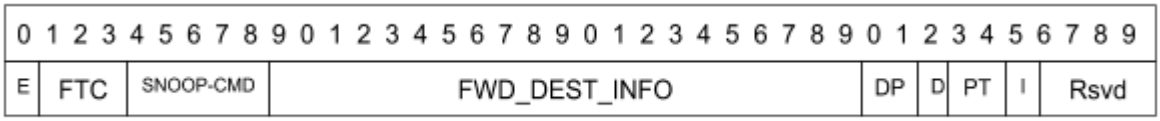
Figure 23: BFD Packet Structure



The following sections provide information about structures of the different headers.

34.12.1 ITMH

Figure 24: ITMH Header



Where:

- E: ITHM base extension exists. Always 1 for BFD.
- FTC: Forward Traffic Class – System level TM traffic class. Configured from the `int_pri` field in the `bfd_endpoint_create()` API, bits 2-4.

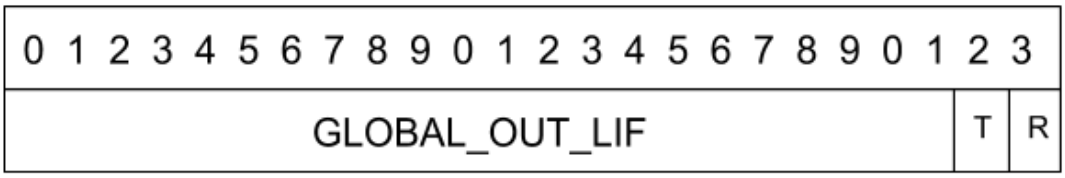
NOTE: Eight profiles available per device to configure FTC and DP, shared with CCM Down MEPs, CCM over MPLS/PWE.

- FWD_DEST_INFO: if `egress_if` field was configured FEC, then this field is the FEC_ID, and the tx_gport field must be invalid (-1). If `egress_if` is an OutLIF, the `tx_gport` field must be valid and this is the value of the tx_gport field.
- DP: drop precedence. Configured from the `int_pri` field in the `bfd_endpoint_create()` API, bits 0-1.

NOTE: Eight profiles are available. See the FTC field description in the previous list.

34.12.2 ITMH Base Extension

Figure 25: ITMH Base Extension



- `GLOBAL_OUT_LIF`: This is the global OutLIF for building link-layer encapsulation. Taken from the `egress_if` field. If the `egress_if` field is a FEC, this value is 0
- **T**: Type of field above. Always 0 for BFD.
- **R**: Reserved

34.12.3 BFD over IPv4/IPv6 One-Hop and Micro BFD

34.12.3.1 IPv4 Header

The header fields are as follows:

- **Version** – 4
- **IHL** – 5
- **Type of Service** – This value is configured by the `ip_tos` API field. Only one global value is allowed per device, first `bcm_bfd_endpoint_create()` sets the value for all subsequent endpoints. It can be modified only when modifying one of the endpoints (using the `BCM_BFD_ENDPOINT_UPDATE` flag) which affects all the IPv4 one-hop and micro BFD endpoints.
- **Total Length** – Length of the packet, measured in bytes, including internet header and data
- **Identification** = 0
- **Flags** = 0
- **Fragment Offset** = 0
- **Time To Live** – 225
- **Protocol** – 17 (UDP)
- **Header Checksum** – Calculated value
- **Source IP Address** – Set by `src_ip_addr` field in `bfd_endpoint_create()` API. This may be extended through extended SIP (see Section 34.12.8, Extended SIP Support).
- **Destination IP Address** – Set by `dst_ip_addr` field in `bfd_endpoint_create()` API.

34.12.3.2 IPv6 Header

The header fields are as follows:

- **Version** = 6
- **Prior** – Configured by the `ip_tos` API field
- **Flow Label** = 0
- **Payload Length** – 8(UDP) + 24(BFD) = 32
- **Next Header** = 17 (UDP)
- **Hop Limit** = configured by `ip_ttl` API field (must be set to 255 for a single hop)
- **Source Address** – Set by the `src_ip6_addr` field in `bfd_endpoint_create()` API
- **Destination Address** – Set by the `dst_ip6_addr` field in `bfd_endpoint_create()` API

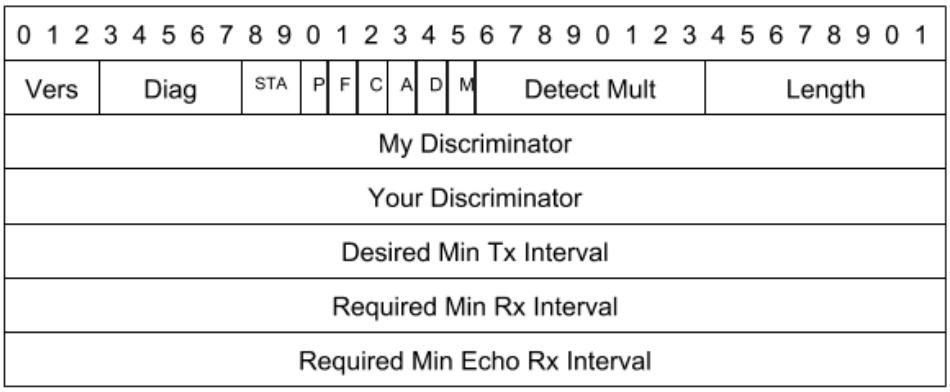
34.12.3.3 UDP Header

The header fields are as follows:

- **SPORT** – Set by `udp_src_port` in `bfd_endpoint_create()` API, with a minimum value of 49,152. One value allowed for BFD over IP and one allowed for BFD over MPLS.
- **DPORT** – 3784 For single-hop, 4784 for multi-hop, 6784 for micro BFD
- **Checksum**:
  - IPv4 – 0x0000
  - IPv6 – Calculated.
- **Length**: The length in bytes of the UDP header and the encapsulated data.

34.12.3.4 BFD PDU

Figure 26: BFD PDU



The header fields are as follows:

- **Vers** – 1
- **Diag** – Configured by the API field `local_diag`. Values 0 to 9 are available.
- **STA** – BFD state. Configured by the API field `local_state`.
- **Flags P, F, C, A, D, M** – Set by the `local_flags` API field. M and A bits must be 0.
- **Detect Mult** – Detection time multiplier. This value is configured by the `local_detect_mult` field.
- **Length** – 24.
- **My Discriminator** – Configured from the `local_discr` field of the API.

  Limitations:

  a. Bit 31 (most significant bit) – May be 0 or 1 per MEP

  b. Bits 30 to 16 – Must have the same value for all BFD endpoints

  c. Bits 0 to 15 – In BCM88670 (Jericho1) compatibility mode, these bits also represent the endpoint ID.

  d. If the endpoint is created WITH_ID (using the `BCM_BFD_ENDPOINT_WITH_ID` flag), limitation c is dropped, and bits 0 to 15 of the *my-discriminator* may be any value on any endpoint. This is valid only in Jer2-system-headers mode and only when the mep-db-full-entry-threshold is set to 0.

  e. If bit 31 is nonzero, then on RX packets, Your-Discriminator field will not be verified in the OAMP. (The classification will still be based on bits 0:18 of incoming packets' Your-Discriminator fields for non-PWE/MPLS-TP endpoints. However, bits 31:19 will not be verified in this case.)
- **Your Discriminator** – Configured by the `remote_discr` API field.
- **Desired Min Tx Interval** – Configured by the `local_min_tx` field.
- **Required Min Rx Interval** – Configured by the `local_min_rx` field.
- **Required Min Echo Rx Interval** – Configured by `local_min_echo field`. Only one value is allowed. It can be modified only when modifying one of the endpoints (using the `BCM_BFD_ENDPOINT_UPDATE` flag), which affects all the endpoints.

34.12.4 BFD over IPv4/IPv6 Multi-Hop

34.12.4.1 IPv4 Header

The header fields are as follows:

- **Version** = 4
- **IHL** = 5
- **Type of Service** – Configured by `ip_tos` API field. This field and Time To Live below are allowed 16 different values as a pair.
- **Total Length** – Length of the packet, measured in bytes, including Internet header and data
- **Identification** – 0.
- **Flags** = 0
- **Fragment Offset** = 0
- **Time To Live** – Configured by `ip_ttl` API field. This field and Type of Service above are allowed 16 different values as a pair.
- **Header Checksum** – A calculated value
- **Source IP Address** – Set by `src_ip_addr` field.
- **Destination IP Address** – Set by `dst_ip_addr` field.

34.12.4.2 IPv6 Header

Where:

- **Version** = 6
- **Prior** – Configured by ip_tos API field
- **Flow Label** = 0
- **Payload Length** – 8 (UDP) + 24 (BFD) = 32
- **Next Header** = 17 (UDP)
- **Hop Limit** = configured by ip_ttl API field
- **Source Address** – Set by src_ip6_addr field in the `bfd_endpoint_create()` API
- **Destination Address** – Set by dst_ip6_addr field in the `bfd_endpoint_create()` API

34.12.4.3 UDP Header

It is the same as single hop (see Section 34.12.3.3, UDP Header).

34.12.4.4 BFD PDU

It is the same as single hop (see Section 34.12.3.4, BFD PDU).

34.12.4.5 BFD o MPLS PHP

In the MPLS network, the original BFD packet is BFDoUDPoIPoMPLSoETH. In the last P node, the incoming packet is BFDoUDPoIPoMPLSoETH. If PHP is enabled, the output packet will be BFDoUDPoIPoETH.

On the PE node, in the RX direction, the incoming packet format is BFDoUDPoIPoETH, which is similar to BFD over MPLS but without the MPLS label.

On the PE node, in the TX direction, the packet sent from PE is the same as BFD over MPLS.

34.12.5 BFD over MPLS

34.12.5.1 LSP Label

Where:

- **Label** – MPLS label. Configured by API field `label`.
- **EXP** – Experimental field, used as class of service. Configured by API field `egress_label`, subfield `exp`.
- **B** – Bottom of Stack (Bos). Always 1 for BFD on MPLS.
- **TTL** – Time To Live. Configured by API field `egress_label`, subfield `ttl`. This field and EXP above are allowed 8 different values as a pair.

34.12.5.2 IPv4 Header

Where:

- **Version** = 4
- **IHL** = 5
- **Type of Service** – Configured by `ip_tos` API field.
- **Total Length** – Length of the packet, measured in bytes, including Internet header and data
- **Identification** – 0.
- **Flags** – 0
- **Fragment Offset** – 0
- **Time To Live** – Always 1 for BFD on MPLS.
- **Header Checksum** – A calculated value
- **Source IP Address** – Set by `src_ip_addr` field in `bfd_endpoint_create()` API.
- **Destination IP Address** – Random value chosen from 127/8 range.

A variation of this encapsulation is supported without the LSP label. In this variation, the LSP label is built by the egress. The ITMH has an FEC pointing to the MPLS label. This is done by using the flag `BCM_BFD_ENDPOINT_FLAGS2_SINGLE_HOP_WITH_RANDOM_DIP` in `bcm_bfd_endpoint_create()`.

Another variation is to add an LSP on top of BFD multi-hop over IPv4/IPv6 built from the OAMP. This may be achieved by setting an FEC pointer on the ITMH.

34.12.5.3 UDP Header

Where:

- **SPORT** – Set by `udp_src_port` in `bfd_endpoint_create()` API, with a minimum value of 49152. One value allowed for BFD over IP and one allowed for BFD over MPLS.
- **DPORT** – Always 3784 for BFD on MPLS.
- **Checksum** – For IPv4: 0x0000. For IPv6: Calculated.
- **Length** – The length in bytes of the UDP header and the encapsulated data.

34.12.5.4 BFD PDU

It is the same as single hop (see Section 34.12.3.4, BFD PDU).

34.12.6 BFD over PWE

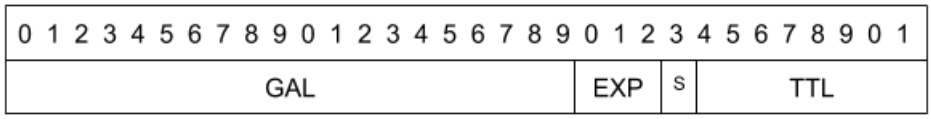
34.12.6.1 PWE Label

Where:

- **Label** – PWE label. Configured by API field label.
- **EXP** – Experimental field, used as class of service. Configured by API field `egress_label`, subfield `exp`. This field and the TTL field below are limited as a pair to 16 different values.
- **S** – Bottom of Stack (Bos). 0 for PWE-GAL, otherwise 1.
- **TTL** – Time To Live. Configured by API field `egress_label`, subfield `ttl`.

34.12.6.2 GAL (Optional)

Figure 27: GAL



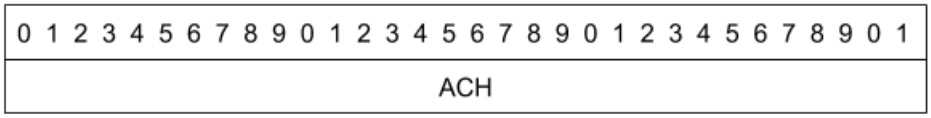
Where:

- **GAL** – General Associated Channel Label. Always 13.
- **EXP** – Experimental field, used as class of service. Value set to 0.
- **S** – Bottom of Stack (Bos). Always 1 for GAL.
- **TTL** – Time To Live. Value set to 64.

NOTE: GAL (General Associated Channel Label) is added if API field type is `bcmBFDTunnelTypeMplsTpCc` or `bcmBFDTunnelTypePweGal`.

34.12.6.3 ACH (Control Word, Optional)

Figure 28: ACH



This channel type value is set using the API `bcm_oam_mpls_tp_channel_type_tx_set()`.

NOTE:

- ACH (associated channel) is added if the API field type is `bcmBFDTunnelTypeMplsTpCc`, `bcmBFDTunnelTypePweGal`, or `bcmBFDTunnelTypePweControlWord`, or if the `flags` field has the bit `BCM_BFD_ENDPOINT_PWE_ACH` (0x40) set and the type is `bcmBFDTunnelTypePweRouterAlert` or `bcmBFDTunnelTypePweTtl`.
- If an additional MPLS/PWE label is required, the `egress_if` field should point to the PWE OutLIF with the same values as those produced by the PWE label in the MEP, which in turn would point to the MPLS label that should be encapsulated. This behavior is similar to the `intf_id` field in `bcm_oam_enpoint_create()` for CCM injection over PWE/MPLS.
- OAMP checks MEP's discriminator through the flexible CRC mechanism. `ipv6_extra_data_index` should be provided with extended data entry.

34.12.7 BFD VCCV Type 1 with IP Encapsulation

This comes with two variations: with a random DIP or with a configurable DIP. In the variation with a configurable DIP, the OAMP will not produce a PWE label.

34.12.7.1 PWE Label

It is the same as with an LSP label in BFD over LSP and only applies when DIP is random.

34.12.7.2 ACH

The channel type is 0x21.

34.12.7.3 IPv4 Header

The TTL will be set to 255. All other fields will be identical to IPv4 header in BFD over LSP.

The destination IP address may optionally be configured if `dst_ip_addr` is nonzero.

34.12.7.4 UDP Header

It is the same as the UDP header in BFD over LSP.

34.12.7.5 BFD PDU

It is the same as single hop (see Section 34.12.3.4, BFD PDU).

To support VCCV type 1 IP format BFD (RFC-5885) with IP encapsulation, create a BFD endpoint with type `bcmBFDTunnelTypeMplsTpCcCv`.

A variation of this type exists where the DIP is a fixed configurable value. When this variation is used, the OAMP does not produce a PWE label and encapsulation should come from the EEDB through the outLIF or FEC.

To support configurable DIP for VCCV type 1 IP format BFD, set the `dst_ip_addr` field. In that case, the IP TTL can be configured as well.

It is also required to create an MPLS port to terminate the PW and ACH of the received packet and set it into the gport field in `bcm_bfd_endpoint_create()`.

The received packet should conform to the following criteria:

- IP-Header.DIP is in the 127/8 range (applies to the variation with a random DIP).
- IP-Header.TTL|Hop-Count == 255 (applies to the variation with a random DIP).
- IP-Header.Protocol is UDP (17)
- UDP-Header.Dst-Port is equal to 3784
- UDP-Header.Src-Port is in the range 49152 to 65535

For a CINT example, refer to `bfd_mpls_run_with_defaults()` with `is_bfd_mpls_tp_cccv=1` and `is_bfd_mpls_tp_cccv_with_fixed_dip` (for a configurable DIP case) in `cint_sand_bfd.c`.

34.12.8 Extended SIP Support

For various BFD over IPv4 endpoints, source IPs can be extended through the `bcm_tunnel_initiator_create()` API by using the following procedure.

1. Call `bcm_tunnel_initiator_create()` with the following settings:
   – `tunnel.type` set to `bcmTunnelTypeIpAnyIn4`
   – `tunnel.sip` set to the required source IP address
   – `tunnel.l3_intf_id` set to the ARP (this field would normally be set on `egress_if` field in `bcm_bfd_endpoint_create()`)
2. The output of this call is the `l3a_intf_id` field of the `bcm_l3_intf_t` struct. This field goes into `egress_if` in `bcm_bfd_endpoint_create()`.

For a CINT example, refer to `bfd_extended_sip` in `cint_sand_bfd.c`.

34.13 BFD Echo

BFD Echo is defined in RFC5880, as an adjunct function to the BFD asynchronous and demand modes. In this function the remote system loops the BFD Echo packets back through its forwarding path, and its main advantage is in testing the forwarding path on the remote system. The BFD echo packet is looped using standard IP forwarding. BFD Echo frames are encapsulated with an IP and UDP header, where the UDP destination port must be 3785.

The Device OAMP supports transmission BFD Echo frames and processing looped-back Echo frames. BFD Echo frames are generated by the OAMP as BFD Control packets. Upon reception, BFD echo packets are identified at the PMF using a specific rule (based on Your-Discriminator), and are sent to the OAMP for monitoring through trap mechanisms. The OAMP verifies the applicable fields, and monitor reception of the packet to determine LoC.

34.13.1 IPv4/IPv6 Header

Same as BFD multi-hop.

34.13.2 UDP Header

Same as BFD multi-hop/single hop, except destination port.

34.13.3 Configuring BFD Echo

1. Add an ACL rule to set trap-Qualifier=BFD.Your-Discriminator[15:0].This is the MEP-ID used by the OAMP to access the MEP-DB. For an example refer to `cint_field_bfd_echo_j2.c`.
2. Call `bcm_bfd_endpoint_create` with:

- `type = bcmBFDTunnelTypeUdp`
- `flags =`
  – `BCM_BFD_ECHO | BCM_BFD_ENDPOINT_IN_HW | BCM_BFD_ENDPOINT_EXPLICIT_DETECTION_TIME | BCM_BFD_ENDPOINT_REMOTE_WITH_ID` (optional) `| BCM_BFD_ENDPOINT_IPV6` (optional).
  – Set the field l`ocal_discr` to a unique identifier per endpoint. This is the value of the transmitted BFD.Your-Discriminator upon which the PMF rule above should be based. This may also determine the MEP-ID (unless the endpoint is created WITH_ID).
  – For IPv6 self-contained entries, provide `ipv6_extra_data_index`.
- All other BFD PDU fields must remain 0.

Fill the IP header fields `bfd_period`, as described above.

34.14 BFD Server

The BFD server application refers to the capability of the OAMP (when configured as a server) to support the BFD endpoint functionality of packets received at another device (client).

BFD packets received and trapped at the client endpoint will be forwarded to the server device, and the server device will forward the packets to the OAMP.

34.14.1 BFD-Server-Client Mode

In this mode, a BFD client endpoint is also defined in the same device as the BFD server in addition to endpoint on the other device. This endpoint is referred to as the *server-client* endpoint and traps the BFD packets and forwards them to the local OAMP.

34.14.2 Configuring the BFD Server

34.14.2.1 Configuring a Trap to the Server Endpoint

A user-defined-trap must be configured in the client device with the `bcm_rx_trap_type_create()` and `bcm_rx_trap_set()` APIs. The destination of the trap must be the server OAMP. The server OAMP port may be found using the API `bcm_port_internal_get()` where the unit should be the server unit and the flags set to `BCM_PORT_INTERNAL_OAMP`. Other fields, such as `flags`, `forwarding_header`, and `egress_forwarding_index` should be set as described in Section 34.3, BFD Traps.

Configure the BFD server endpoint by using the `bcm_bfd_endpoint_create` API with the `BCM_BFD_ENDPOINT_IN_HW` and `BCM_BFD_ENDPOINT_HW_ACCELERATION_SET` flags. `remote_gport` must be set to the trap code described previously.

All endpoint types and flags may be used, such as:

- `BCM_BFD_ENDPOINT_MULTIHOP`
- `BCM_BFD_ENDPOINT_IPV6`
- `BCM_BFD_ECHO`
- Seamless BFD, Micro BFD, and so on

All other fields used for creating accelerated endpoints may be used.

34.14.2.2 Configuring the BFD Client

Configure the BFD client Endpoint using the ID of the server endpoint.

Use the `bcm_bfd_endpoint_create` API with the `BCM_BFD_ENDPOINT_WITH_ID` and `BCM_BFD_ENDPOINT_IN_HW` flags and the server endpoint's ID to couple the client endpoint to the server endpoint. `remote_id` is required when using client endpoints with the same ID to different server devices. Bits 24 to 28 of the endpoint ID will be set to signify the server device's profile information of the client ID. In other words, the endpoint will be created with ID X but the actual endpoint ID will be 0xY000000 + X, where *Y* is the server device profile.

In addition, set `remote_gport` with the server's OAMP as the destination.

All endpoint types and flags may be used, such as:

- BCM_BFD_ENDPOINT_MULTIHOP
- BCM_BFD_ENDPOINT_IPV6
- BCM_BFD_ECHO
- Seamless BFD, Micro BFD, and so on

Set the client trap using the server unit's OAMP as destination port. This will forward trapped BFD packets to the server's OAMP. Place the trap code in the remote_gport field.

34.14.2.3 Configuring BFD Server-Client Endpoint

The BFD Server-Client Endpoint resides on the server's device and traps BFD packets to the OAMP. It should be configured with the BCM_BFD_ENDPOINT_IN_HW flag, where remote_gport is configured with the local OAMP as the destination port.

All endpoint types and flags may be used, such as:

- BCM_BFD_ENDPOINT_MULTIHOP
- BCM_BFD_ENDPOINT_IPV6
- BCM_BFD_ECHO
- Seamless BFD, Micro BFD, and so on

34.14.3 Application Reference

An example cint with the full configuration flow over two units is available in cint_bfd_over_lag.c.

34.15 BFD Default Endpoints

In addition to standard endpoints, there are also *default* endpoints. This is a resource shared with the OAM feature. For OAM, there are four default ingress endpoints and four default egress endpoints. The ingress endpoints can also function as default BFD endpoints.

BFD packets are normally classified either by the *your-discriminator* field (if sent to a single-hop, multi-hop or micro-BFD endpoint) or by a LIF (for MPLS or PWE endpoints). If these classifications fail, the packet can still be handled by one of the four default endpoints, which are defined by a LIF/RIF property. The only parameters for default BFD endpoints are trap, trap strength, and snoop strength (provided in the field remote_gport); thus, the only actions possible as a result of BFD default endpoint classification are trapping and snooping.

34.15.1 SOC Properties

None

34.15.2 Configuration Flow

1. On a given LIF/RIF, use the bcm_port_control_set() API to set the OAM Default Profile to any value between 0 and 3.
   a. Set port to the LIF/RIF
   b. Set the type to bcmPortControlOamDefaultProfile
   c. Set the value to the profile assigned to the LIF.

2. Use the `bcm_bfd_endpoint_create()` API to create an endpoint on the LIF profile.
   a. ID should be set to `BCM_BFD_ENDPOINT_DEFAULT0` to `BCM_BFD_ENDPOINT_DEFAULT4`, depending on the value in Step c in the previous step.
   b. `remote_gport` should be set to the trap code along with snoop/trap strength.

34.15.3 Shell Commands

None

34.15.4 Application Reference

An example can be found in `bfd_default_mep_over_lif_example()` in `cint_sand_bfd.c`.

34.16 Micro BFD

Micro BFD is defined in RFC7130. This particular protocol was created to ensure connectivity inside a link aggregation group (LAG), which is defined in IEEE802.1AX, “provides mechanisms to combine multiple physical links into a single logical link.” Thus, micro-BFD sessions are defined as “Asynchronous mode BFD sessions on every LAG member link.” BFD control packets for micro-BFD sessions are IP/UDP encapsulated as defined in RFC5881, but with UDP destination port 6784.

The device OAMP supports transmission and reception of micro-BFD session frames.

34.16.1 Micro-BFD Session Ethernet Details

According to RFC-7130, the DMAC 01-00-5E-90-00-01 “*must* be used for the initial BFD packets of a micro-BFD session when in the Down, AdminDown, and Init states. When a micro-BFD session is changing into the Up state, the first `bfd.DetectMult` packets in the Up state *must* be sent with the dedicated MAC.”

To support this requirement on the RX side, it is recommended to configure DMAC as an additional my-MAC address of all VSIs, see Section 22.5, My-MAC Enhancements (VRRP and Multiple My-MAC).

For the TX side, the DMAC may be controlled through the `egress_if` field in `bcm_bfd_endpoint_create()`.

34.16.2 SOC Properties

None

34.16.3 Configuration Flow

Configuration of an accelerated micro-BFD endpoint is similar to single-hop endpoint, both over IPv4 or IPv6, only with the flag `BCM_BFD_ENDPOINT_MICRO_BFD` set in the `flags` field.

34.16.4 Shell Commands

None

34.16.5 Application Reference

An example for IPv4 is found in `bfd_ipv4_example()` in `cint_sand_bfd.c`

An example for IPv6 is found in `bfd_ipv6_run_with_defaults()` in `cint_sand_bfd_ipv6.c`

34.17 Seamless BFD

Seamless BFD is defined in RFC7880 and RFC7881. It features a state machine that is more simple than the classical BFD state machine to enable faster setup of continuity tests. It uses the same PDU format as classical BFD.

Network nodes have two roles in Seamless BFD: initiator and reflector. The initiator initiates transmission of an SBFD packet towards the reflector, and the reflector reflects the packet back to the initiator. Only one SBFD reflector endpoint is supported. For more details about the seamless BFD feature, refer to RFC7880 and RFC7881.

The device OAMP supports transmission and reception of Seamless-BFD session frames.

34.17.1 SOC Properties

Reflector:

```
ucode_port_<rcy_port_num>=RCY.1
tm_port_header_type_in_<rcy_port_num>= RCH_0
tm_port_header_type_out_<rcy_port_num>=ETH
```

34.17.2 Configuration Flow

Initiator

- Call `bcm_bfd_endpoint_create` with:
  - `type` – `bcmBFDTunnelTypeUdp` (SBFDoIPV4) or `bcmBFDTunnelTypeMpls` (SBFDoMPLS)
  - `flags` – `BCM_BFD_ECHO` is not supported for seamless BFD initiator
  - `flags2 = BCM_BFD_ENDPOINT_FLAGS2_SEAMLESS_BFD_INITIATOR`
  - `local_discr` – Set the field `local_discr` to a unique identifier per system.
  - `src_ip_addr` is required for IPv4, and `src_ip6_addr` must be NULL
  - `src_ip6_addr` is required for IPv6, and `src_ip4_addr` must be NULL
  - `udp_src_port` is required. It must be the same value for all initiator endpoints. The value may be any value from 0 to 65536, except 7784.
  - `local_flags` – Demand bit must be set (value 2).
  - `local_min_rx=0`
  - `local_min_echo=0`
  - `remote_discr` must be nonzero

  Otherwise, all classical BFD EP rules apply (accelerated, `remote_gport` and so on.)

Reflector

- Only one reflector endpoint can be created (one for MPLS and one for IPv4 or IPv6).
- Call `bcm_bfd_endpoint_create` with:
  - `type` – `bcmBFDTunnelTypeUdp` (SBFDoIPV4 or SBFDoIPV6) or `bcmBFDTunnelTypeMpls` (SBFDoMPLS).
  - `flags` – `BCM_BFD_ENDPOINT_IPV6` (SBFDoIPV6 only).

  - flags2 – BCM_BFD_ENDPOINT_FLAGS2_SEAMLESS_BFD_REFLECTOR and BCM_BFD_ENDPOINT_FLAGS2_USE_MY_DIP_DESTINATION (SBFDoIPV6 only).
  - local_min_echo and local_min_rx – A unique value per system can be defined.
  - src_ip_addr can be configured for SBFDoMPLS reflector endpoint.
  - remote_gport must be the trap code with trap SBFD packet to recycle port.
    - Create TRAP to trap SBFD packet to recycle port through bcm_rx_trap_type_create and bcm_rx_trap_set.
  - local_state – A unique value per system can be defined.
  - All other BFD PDU fields must remain 0.
- Create RCH encap through bcm_l2_egress_create with:
  - flags = BCM_L2_EGRESS_RECYCLE_HEADER.
  - recycle_app = bcmL2EgressRecycleAppDropAndContinue.
- Configure recycle port term_context_port_profile through bcm_port_control_set with bcmPortControlRecycleAppSbfdReflector.
- Create SBFD reflector encap through bcm_switch_reflector_create with bcmSwitchReflectorSbfdIp4, bcmSwitchReflectorSbfdIp6, or bcmSwitchReflectorSbfdMpls.
- Use ACL to modify TTL to 255. TTL could be decreased from 255 to 254, which is not a detail covered by the RFC.
- SBFD reflector is supported in interop mode. Add the L3 route for the second pass in this mode.
- SBFD reflector over IPV6 packet is identified through My-BFD-DIP. Set a special BFD_MY_DIP_DESTINATION value by using the bcm_switch_control_set API with type=bcmSwitchBfdMyDipDestination. Add the L3 route from the intended destination IP to the BFD FEC by using the bcm_l3_route_add API.

34.17.3 Shell Commands

None

34.17.4 Application Reference

Initiator

An example for an SBFD initiator can be found in cint_sand_bfd.c. When setting is_s_bfd_init_ep=1 before calling bfd_ipv4_run_with_defaults() or bfd_mpls_run_with_defaults() results in a seamless BFD initiator endpoint (IPv4 and MPLS, respectively). For an IPv6 example, refer to cint_sand_bfd_ipv6. After setting is_s_bfd_init_ep=1, call the functions bfd_ipv6_service_init() and bfd_ipv6_run_with_defaults().

Reflector

An example for SBFD reflector can be found in sbfd_reflector_endpoint_create() in cint_sand_bfd_sbfd_reflector.c.

An example for SBFD reflector with SRV6 can be found in sbfd_reflector_ipv6_with_srv6_run() in cint_sand_bfd_sbfd_reflector.c.

34.18 BFD over VXLAN

Different combinations of BFD over VXLAN can be transmitted. If a VXLAN encapsulation FEC is defined, it can be used in the egress_if field when creating or modifying a BFD endpoint over UDP, including single-hop or multi-hop, and IPv4 or IPv6.

Only IPv4 Underlay is supported with up to one VLAN tag for the underlay L2 header.

Accelerated BFD IPv4 over VXLAN also supports extended SIP.

34.18.1 SOC Properties

None

34.18.2 Configuration Flow

Any BFD over UDP endpoint can receive a VXLAN packet, whether IPv4 or IPv6, single-hop or multi-hop.

For an accelerated BFD endpoint to transmit a VXLAN packet, the endpoint must be UDP (IPv4 or IPv6, single-hop or multi-hop) and the destination configured must be a FEC into a VXLAN encapsulation. See Section 34.10, Creating BFD Endpoints, for information about how to configure the destination as a FEC.

34.18.3 Shell Commands

None

34.18.4 Application Reference

For an example on how to configure a BFD over UDP endpoint with a destination VXLAN FEC, see the following:

- IPv4 – File: `cint_sand_bfd.c`.
  Set `use_vxlan_fec=1` before calling the function `bfd_ipv4_run_with_defaults()`.
- IPv6 – File: `cint_sand_bfd_ipv6.c`.
  Set `use_vxlan6_fec=1` before calling the function `bfd_ipv4_run_with_defaults()`.

34.19 BFD Endpoint Destruction

To destroy a single BFD endpoint, use `bcm_bfd_endpoint_destroy()`. The endpoint ID used as an argument is the value returned in the id field of the `bcm_bfd_endpoint_info_t` struct used to call `bcm_bfd_endpoint_create()`.

To destroy all BFD endpoints in a unit, call `bcm_bfd_endpoint_destroy_all()`. It works by iterating through all existing BFD endpoints and calingl `bcm_bfd_endpoint_destroy()` for each one.

Chapter 35: Ethernet VPN (EVPN)

35.1 Introduction

EVPN is an L2 VPN application that is configured over an MPLS infrastructure. The underlying forwarding method is Ethernet bridging. An Ethernet packet is bridged into an encapsulation which includes a label (EVPN) over an MPLS stack. The packet travels across the MPLS core and is bridged outwards following a termination of the EVPN label and the underlying MPLS stack.

For broadcast, unknown, and multicast traffic (BUM), the incoming Ethernet packet is replicated, and replications going into the core are encapsulated with a multicast label (IML). Replications that are destined through the MPLS core to switches (PEs) that are connected to the packet origin Ethernet segment should have an indication of the origin in the form of another label below the IML, containing the Ethernet Segment Identifier (ESI). The egress PE should use this identifier to filter replications destined to the origin Ethernet segment.

For compatibility reasons, this application is configured using the `bcm_mpls_tunnel_switch_create` (for EVPN termination) and `bcm_mpls_tunnel_initiator_create` (for EVPN/IML encapsulation).

The ESI encapsulation and termination are done using the API `bcm_mpls_esi_encap_add`, which this chapter also covers.

This chapter describes how to set up the EVPN applications using these APIs.

Before reading this chapter, read the following chapters:

- Chapter 21, Ethernet Bridge and Chapter 22, IP Router to get an overview of the basic L2 and L3 objects.
- Chapter 25, MPLS LER Encapsulation. This section provides information regarding the encapsulation of MPLS labels.
- Chapter 24, MPLS LER Termination. This section provides information regarding the termination of MPLS labels.

35.2 Application Configuration Checklist

35.2.1 Encapsulation (Ingress PE)

- Add EVPN and IML encapsulations
- Add ESI encapsulations
- Create ARP entries (l3_egress) and out-AC.
- Create MPLS label encapsulations pointing to the ARP entries
- Create FEC pointing to the port and MPLS tunnel
- Setup Split-Horizon
- Add IML to flooding multicast groups
- Add MACT entries / Cross connections

35.2.2 Termination (Egress PE)

- Configure IML range
- Configure ESI filter ingress FP application
- Configure ESI filter egress FP application
- Add EVPN termination
- Add IML termination
- Add ESI + forbidden port to egress FP DB
- Add ESI to egress FP DB
- Add MACT entries / Cross connections

35.3 EVPN and Inclusive Multicast Label (IML) Encapsulation

EVPN and IML MPLS labels are used to implement RFC 7432—BGP MPLS-based Ethernet VPN. EVPN label refers to L2-VPN objects, used for unicast traffic in the EVPN network core. Inclusive Multicast Label (IML) refers to L2-VPN objects, used for multicast through the EVPN network core.

The device allows encapsulating a packet with an EVPN or IML label, usually at the bottom of the stack. IML labels trigger a lookup of ESI labels to signal the originating ES to a forwarder peer PE. The IML object provides a class-ID field that is used later as part of the ESI lookup key. The user is responsible for the management and allocation of the values for the field. The available value range for the field is the same as the VLAN domain. For more on ESI lookup, see Section 35.4, ESI Encapsulation.

35.3.1 SOC Properties

None

35.3.2 Configuration Flow

35.3.2.1 EVPN Label Encapsulation

To create an EVPN label, call API `bcm_mpls_tunnel_initiator_create(unit, 0, 1, &label_info)`:

- `label_info.label` – The `label_value`
- `label_info.flags`:
  - `BCM_MPLS_EGRESS_LABEL_EVPN` – Must be set
  - `BCM_MPLS_EGRESS_LABEL_CONTROL_WORD` – For control word (CW) encapsulation
  - `BCM_MPLS_EGRESS_LABEL_ENTROPY_ENABLE` – For flow label encapsulation
- `label_info.encap_access` – Can be either `bcmEncapAccessTunnel1` or `bcmEncapAccessTunnel2`.
- For information about other fields, see Section 23.4, Egress MPLS Tunnel

35.3.2.2 IML Label Encapsulation

To create an IML egress label object, call API `bcm_mpls_tunnel_initiator_create(unit, 0, 1, &label_info)`:

- `label_info.label` – label_value
- `label_info.flags`:
  - `BCM_MPLS_EGRESS_LABEL_EVPN` – Must be set
  - `BCM_MPLS_EGRESS_LABEL_CONTROL_WORD` – For control word (CW) encapsulation
  - `BCM_MPLS_EGRESS_LABEL_ENTROPY_ENABLE` – For flow label encapsulation
- `label_info.encap_access` – Can be either `bcmEncapAccessTunnel1` or `bcmEncapAccessTunnel2`.
- `label_info.flags2` – Set it with `BCM_MPLS_EGRESS_LABEL_FLAGS2_ESI_INTF_NAMESPACE` in virtual segment cases.
- For information about other fields, see Section 23.4, Egress MPLS Tunnel.

ESI lookup uses the IML encapsulation entry's class-ID. To set it, call with IML object:

```
int gport; BCM_L3_ITF_LIF_TO_GPORT_TUNNEL(gport, label_info.tunnel_id);
bcm_port_class_set(unit, gport, bcmPortClassEgress, <class_id>);
```

NOTE: The default value for the `encap_access` field is `bcmEncapAccessInvalid`. It is considered a valid value, with the same effect as `bcmEncapAccessTunnel1`. When using `bcm_mpls_tunnel_initiator_get` the `bcmEncapAccessInvalid` value is always returned, even if `bcmEncapAccessTunnel1` is set.

35.3.3 Shell Commands

None

35.3.4 Application Reference

EVPN Label encapsulation

Example application defining EVPN label encapsulation for unicast traffic

- **Type:** Application reference
- **Path**: `$SDK/src/examples/dnx/cint_dnx_evpn_basic.c`
- **Function**: `evpn_service_egress_config`

IML Label encapsulation

Example application defining IML label encapsulation for multicast traffic

- **Type:** Application reference
- **Path**: `$SDK/src/examples/dnx/cint_dnx_evpn_basic.c`
- **Function**: `iml_encap_add`

35.4 ESI Encapsulation

ESI Labels are MPLS labels, added under IML label in the MPLS stack, to identify the packet's origin Ethernet segment (ES) to a PE, which is a forwarder to that same ES. These labels are downstream assigned (i.e the receiving PE published them, to identify the ES to which he is forwarding packets). This means that each peer PE must get his own ESI labels depending on the origin ES.

The device supports egress DB, from which ESI labels may be retrieved when encapsulating an IML label. Each peer PE, which is dual homed to the device, must be assigned an identifier (class-ID) that will be used as part of the key to this DB. The other part is the source system port or source virtual port (for example, the source VLAN port), which should be unique per-ES.

For information about how to set the peer class-ID on the IML encapsulation object, see Section 35.3, EVPN and Inclusive Multicast Label (IML) Encapsulation.

NOTE: If the native AC in the encapsulation stack is added explicitly, ESI is not supported.

35.4.1 SOC Properties

None

35.4.2 Configuration Flow

To add an entry to the egress ESI DB, call `bcm_mpls_esi_encap_add(unit, &esi_info)`:

- `esi_info.flags` – Use 0 for general a Ethernet segment or `BCM_MPLS_ESI_INTF_NAMESPACE` for a virtual segment.
- `esi_info.esi_label` – ESI label value
- `esi_info.src_port` – Source system port , for example.
  `BCM_GPORT_SYSTEM_PORT_ID_SET(esi_info.src_port, <system_port>);`
- `esi_info.out_class_id = peer_class_id`

35.4.3 Shell Commands

None

35.4.4 Application Reference

Example application defining ESI encapsulation

- **Type:** Application reference
- **Path**: `$SDK/src/examples/dnx/cint_dnx_evpn_basic.c`
- **Function**: `evpn_service_egress_config`

35.5 ESI Encapsulation in IOP Mode

In BCM88670 (Jericho1) system-headers mode (IOP mode), the ESI labels are upstream assigned (that is, the transmitting PE published them, to identify the ES from which the packets are forwarded.) This means that all peer PEs share the same ESI labels for the same ES.

Except for the egress DB for ESI encapsulation in DNX devices, users must set the ESI label and the PE-ID (of the Designated Forwarding PE of the original ES) to the port which the ES is connected to. This information - {PE-ID, ESI label} is copied to the system headers and carried to the egress pipeline for ESI label encapsulation for Jericho1 devices ignored for DNX devices).

35.5.1 Shell Commands

None

35.5.2 Configuration Flow

To set the ESI label and DF PE-ID per ingress port, call `bcm_port_class_set(unit, port, pclass, class_id)`:

- `port` – The port the ES segment is attached
- `pclass` – `bcmPortClassFieldIngressPMF1PacketProcessingPortGeneralData`, `bcmPortClassFieldIngressPMF1PacketProcessingPortGeneralDataHigh`
- `class_id` – Data constructed by ESI label and PE-ID.

NOTE: Any of the two `plcass` values can set up to 32 bits of data. However, ESI label and PE-ID totally occupy 40 bits. So, call the API twice, one with 32 LSBs of {PE-ID, ESI label} and another with the 8 MSBs of it.

35.6 Field Processor for ESI Label Copying in IOP Mode

To copy the {PE-ID, ESI label} setting in the ingress port to the system header, call the Field Processor. To get aligned with Jericho1 devices, the user-defined header (UDH) carries this information. For this reason, enable UDH-related SOC properties to define the size of UDH as done in Jericho1 devices.

A copy is performed on the InLIF or in-Port selected by the user. Mark the InLIFs with a special, dedicated field-processor indicator (for example, mark a special value to the InLIF profile for the field processor to identify). The information in this section assumes the marking has occurred.

35.6.1 SOC Properties

The SOC properties are as follows:

- `field_class_id_size_0=8`
- `field_class_id_size_1=8`
- `field_class_id_size_2=16`
- `field_class_id_size_3=8`

35.6.2 Configuration Flow

To set the field processor identification to InLIF profile, call call `bcm_port_class_set(unit, port, pclass, class_id)`:

- `port` – The gport that represent the InLIF
- `pclass` – `bcmPortClassFieldIngressVport`
- `class_id` – A value defined by user for field processor identification.

35.6.3 Application Reference

Example application Field processor for copying ESI to UDH

- **Type**: Application reference
- **Path**: `$SDK/src/examples/sand/cint_field_evpn.c`
- **Function**: `field_jr2_comp_evpn_db(unit, presel_id, forwarding_type)`

35.7 ARP with VLAN Translation Entries for EVPN

EVPN encapsulations, IML encapsulations in particular, require that the outer ETH VLAN translation information comes from the ARP encapsulation.

For information on how to create and configure ARP entries with AC information, see Section 22.7, L3 Egress Object: Egress-ARP (Next-Hop).

35.7.1 Application Reference

Example application defining ARP + VLAN-translation entry

- **Type:** Application reference
- **Path**: `$SDK/src/examples/dnx/cint_dnx_evpn_basic.c`
- **Function**: `evpn_mpls_arp_plus_ac_configure`

35.8 Split-Horizon Filtering

Split Horizon filtering refers to filtering (dropping) copies of forwarded/flooded packets that arrive from the EVPN core network and are destined back out to the same core.

In the BCM SDK, each core is identified by a *Network Group*, and the device can be configured to drop copies from one network group to another group or the same group (map <In-NWK-GRP, Out-NWK-GRP> to a drop indication).

Generally, the out-network-group is taken from the EVPN/IML OutLIF. Instead, it is taken from the native VLAN translation information. The user should configure all native VLAN translation entries with the required network group and, in addition, it is recommended to create default native VLAN translation action per EVPN core, with the network-group-id relevant to that core.

This is done using the standard VLAN port and VLAN translation APIs. See Section 20.2.4, Configure Split Horizon for Egress L2 VPN Tunnels (PWE, VXLAN, EVPN).

35.8.1 Application Reference

Example application defining default native VLAN translation for EVPN

- **Type:** Application reference
- **Path**: `$SDK/src/examples/dnx/cint_dnx_evpn_basic.c`
- **Function**: `evpn_service_egress_config`

35.9 Add IML LIF to Multicast Group

To add an IML with the carrying MPLS tunnel, ARP, and AC to a multicast group (for any BUM traffic), two options exist:

- Use a single OutLIF that points to the beginning of the encapsulation chain (IML, MPLS, ARP + AC)
- Use a handle that would be resolved in egress to several different OutLIFs (using PP multicast DB).

This section describes the second option. The first option is based on standard MC Group management.

For more information about PP multicast see Chapter 4, MC Encapsulation Extension.

35.9.1 SOC Properties

None

35.9.2 Configuration Flow

Create egress mapping from a handle to several OutLIFs:

```
int mc_ext_id = 0;
int outlifs[2];
outlifs[0] = <iml_tunnel_id>;
outlifs[1] = <mpls_tunnel_id>;
bcm_multicast_encap_extension_create(unit, 0, &mc_ext_id, 2, outlifs);
```

The returned `mc_ext_id` should be added to the required multicast groups.

35.9.3 Shell Commands

None

35.9.4 Application Reference

Example application adding IML encapsulation to an MC group:

- **Type:** Application reference
- **Path**: `$SDK/src/examples/dnx/cint_dnx_evpn_basic.c`
- **Function**: `iml_encap_add`

35.10 IML Range

An ingress IML label may have an ESI label under it in the MPLS stack, or the IML label could be at the bottom of the stack itself. The indication whether an ESI exists comes from the BOS bit of the IML label while it is terminating. For this purpose, IML labels should be identified before they are resolved to a LIF so that they can be resolved together with an accompanying ESI label.

Define the IML label value range. Labels with values in that range are matched by their label values and BOS bit values before they are resolved to LIFs so they can be resolved together with an accompanying ESI label.

This range of labels should not be used for non-IML labels. It should be defined at the start of the application to identify IML labels correctly. It is the responsibility of the user to avoid configuring the termination of non-IML label termination with values in the IML range.

If the range needs to be changed, it is the user's responsibility to make sure IML termination objects that will become out of range are deleted prior to the range change, and non-IML termination objects that will become in IML range after the change, modified as well.

Neglecting to do so by the user may render some objects unreachable for deletion and cause resource leakage and collisions.

35.10.1 SOC Properties

None

35.10.2 Configuration Flow

Configure MPLS label range for IML labels:

```
bcm_mpls_range_action_t action;
action.flags = BCM_MPLS_RANGE_ACTION_EVPN_IML;
bcm_mpls_range_action_add(unit, <range_min>, <range_max>, &action);
```

35.10.3 Shell Commands

None

35.10.4 Application Reference

Example application defining IML termination range setup

- **Type:** Application reference
- **Path**: `$SDK/src/examples/dnx/cint_dnx_evpn_basic.c`
- **Function**: `evpn_service_ingress_config`

35.11 Field Processor Application for ESI Filtering

To filter traffic on an egress PE to a packet originating ES, use a field processor application. The reference application demonstrates the usage of a user-defined header built in the ingress device (of the egress PE) and a priority-based DB in the egress device to drop:

- Copies of a MC packet that are destined to a forbidden port (per-ESI value) and
- All copies of a packet arriving with an undefined ESI label value.

The general idea is to allow more than *dual* homing using this application, since more than a single forbidden port can be defined per-ESI.

The example uses generic FP APIs and the user may choose to implement a different solution based on FP resource and application requirements. See Chapter 11, Field Processor for more information.

35.11.1 SOC Properties

None

35.11.2 Configuration Flow

See the reference application.

35.11.3 Shell Commands

None

35.11.4 Application Reference

Example for the previously described FP applications:

- **Type:** Application reference
- **Path**: `$SDK/src/examples/dnx/field/cint_field_evpn_esi.c`

NOTE: This application assumes either all incoming IMLs have a CW or all incoming IMLs do not have a CW. To support CW for some IMLs and no CW for others, this reference can be used twice with different values for `bcmFieldQualifyInVportClass`.

35.12 Field Processor for ESI Filtering in BCM88670 (IOP) Mode

The field processor is also advised for ESI filtering even in BCM88670 (JR1) mode as in Jericho1 devices. The application has a few differences from BCM88690 (JR2) mode. Generally, the solution aligns with Jericho1 devices.

The filter is implemented in two parts:

- ESI checker – Check if there is an ESI label as expected in the packets
- ESI filter – Drop the packet matches with a combination of ESI label and forbidden out port. This is to prevent the packet getting back to the original ES.

The ESI checking is done in the ingress FP. Even so, the InLIF profile should indicate whether the ESI label is expected in the packet. That means the user should set an identification to the InLIF profile in the IML LIF with ESI. For more information, see Section 35.14, IML Label Termination.

Define at least two actions in the FP:

- Set a forbidden out port to learn extension in system header if the ESI label matches.
- Drop the packet if the ESI label does not match.

Add all in-use ESI labels and their forbidden out ports to this FP. Finally, add a default entry for discarding the packets that do not have the ESI label as expected.

ESI filtering is done in the egress FP. The same solution as in Jericho1 is advised. For details, refer to the EVPN section in the BCM88670 PP user manual (88670-PG1xx).

35.12.1 SOC Properties

```
bcm886xx_pph_learn_extension_disable = 0
```

35.12.2 Configuration Flow

See the reference application.

35.12.3 Shell Commands

None

35.12.4 Application Reference

Example for the ESI checker FP applications

- **Type:** Application reference
- **Path:** `$SDK/src/examples/sand/cint_field_evpn.c`
- **Function:** `field_jr2_comp_evpn_esi_db(unit, presel_id)`

Example for the ESI filter FP applications

- **Type:** Application reference
- **Path:** `$SDK/src/examples/sand/cint_field_evpn.c`
- **Function:** `field_jr2_comp_evpn_egress_filter_db(unit, presel_id)`

35.13 EVPN Label Termination

EVPN and IML MPLS labels are used to implement RFC 7432 – BGP MPLS-based Ethernet VPN. EVPN label refers to L2-VPN objects used for unicast traffic in the EVPN network core.

The device can terminate EVPN labels at the bottom of the MPLS stack.

Ingress VLAN translation can be performed after EVPN label termination. The Native-AC information used by the VLAN translation is always classified with interface namespace. For more information about VLAN translation, see Chapter 16, VLAN Editing Mechanism.

35.13.1 SOC Properties

None

35.13.2 Configuration Flow

Configure EVPN label termination by calling `bcm_mpls_tunnel_switch_create(unit, &evpn_info)`:

- `evpn_info.action = BCM_MPLS_SWITCH_ACTION_POP;`
- `evpn_info.flags = BCM_MPLS_SWITCH_NEXT_HEADER_L2;`
- `evpn_info.flags2`
  - `BCM_MPLS_SWITCH2_CROSS_CONNECT` – For P2P
  - `BCM_MPLS_SWITCH2_CONTROL_WORD` – For CW termination
  - `BCM_MPLS_SWITCH_ENTROPY_ENABLE` – For flow label
- `evpn_info.label = <label_value>;`
- `evpn_info.vpn = <vpn_id>; // 0 for P2P`
- For information about other fields, see Section 24.3, Ingress MPLS Termination Object.

NOTE: It is not possible to change the service type (P2P/MP) of an existing EVPN label. That is, labels created with the `BCM_MPLS_SWITCH2_CROSS_CONNECT` flag set, cannot clear it using REPLACE APIs and the other way around.

Unknown-DA MACs: To set flooding group offset call `bcm_port_flood_group_set(unit, port, flags, flood_groups)`.

- Flags are not used currently. For `flood_groups`, it is defined as follows:

```
typedef struct bcm_port_flood_group_s {
      bcm_gport_t unknown_unicast_group;
      bcm_gport_t unknown_multicast_group;
      bcm_gport_t broadcast_group;
} bcm_port_flood_group_t;
```

- Members in the structure are used as an offset to flooding groups for unknown unicast, unknown multicast and broadcast packets. The value used for them is gport that can be:
  - `BCM_GPORT_BLACK_HOLE` – Drop packets on specific InLIF or in-port. See Chapter 2, Ports and Generalized Ports, for additional configuration required for black hole.
  - `BCM_GPORT_TYPE_MCAST` – The Multicast-ID is the offset to flooding groups for Unknown packets.
  - `BCM_GPORT_TYPE_TRAP` – Flooding groups for unknown packets are destined to configured trap.
  - `BCM_GPORT_INVALID` – Keep the flooding group unchanged.

35.13.3 Shell Commands

None

35.13.4 Application Reference

Example application defining EVPN label termination

- **Type:** Application reference
- **Path**: `$SDK/src/examples/dnx/cint_dnx_evpn_basic.c`
- **Function**: `evpn_service_ingress_config`

35.14 IML Label Termination

IML labels are used for BUM traffic in the EVPN core and should be at the bottom of the MPLS stack (BOS), or one above the bottom if the BOS is an ESI label.

Ingress VLAN translation can be performed after EVPN label termination. The Native-AC information used by the VLAN translation is always classified with interface namespace. For more details about VLAN translation, see Chapter 16, VLAN Editing Mechanism.

35.14.1 IML Label Termination – Handling Two Logical Interfaces

Two InLIFs are required to handle the separate cases (with and without ESI). One IML LIF without ESI is similar to an EVPN label LIF. Another IML LIF with ESI is also for terminating the ESI label but can be configured to trigger the FP application for ESI filtering based on the value of the ESI label.

In both cases, the LIF classification is based on the IML label value and its BOS bit value. This is different from normal MPLS label classification, in which the IML label value should be in a user-defined range (see Section 35.10, IML Range).

Both of the LIFs (with and without ESI) can be created, or only one can be created to handle incoming traffic.

35.14.2 SOC Properties

None

35.14.3 Configuration Flow

Use the following steps to support IML labels termination by handling two logical interfaces handling:

- To create IML label termination with ESI, call:
  - `bcm_mpls_tunnel_switch_create(unit, &iml_info):`
  - `iml_info.action = BCM_MPLS_SWITCH_ACTION_POP;`
  - `iml_info.flags`
    - `BCM_MPLS_SWITCH_EVPN_IML` – Multicast label indicator, mandatory
    - `BCM_MPLS_SWITCH_ENTROPY_ENABLE` – For flow labels
  - `iml_info.flags2 =`
    - `BCM_MPLS_SWITCH2_CROSS_CONNECT` – For P2P
    - `BCM_MPLS_SWITCH2_CONTROL_WORD` – For CW termination
  - `iml_info.label = <label_value>;`
  - `iml_info.vpn = <vpn_id>;`
  - Other fields, see Section 23.3, Ingress MPLS Termination Object.

NOTE: It is not possible to change the service type (P2P/MP) of an existing IML label. That is, for labels created with the `BCM_MPLS_SWITCH2_CROSS_CONNECT` flag set, it is not possible to clear it using `REPLACE` APIs and vice versa.

- To create IML label termination without ESI call:

  `bcm_mpls_tunnel_switch_create(unit, &iml_info):`

  - `iml_info.action = BCM_MPLS_SWITCH_ACTION_POP;`
  - `iml_info.flags =`
    - `BCM_MPLS_SWITCH_EVPN_IML` – Multicast label indicator, mandatory
    - `BCM_MPLS_SWITCH_EXPECT_BOS` – Expect BOS indicator, mandatory
    - `BCM_MPLS_SWITCH_ENTROPY_ENABLE` – For flow labels
  - `iml_info.flags2 =`
    - `BCM_MPLS_SWITCH2_CONTROL_WORD` – For CW termination
  - `iml_info.label = <label_value>;`
  - `iml_info.vpn = <vpn_id>; // 0 for P2P`
  - Other fields, please see MPLS Termination object.

It is not possible to change the service type (P2P/MP) of an existing IML label. That is, labels created with the `BCM_MPLS_SWITCH2_CROSS_CONNECT` flag set cannot clear it using `REPLACE` APIs and vice versa.

`BCM_MPLS_SWITCH_ENTROPY_ENABLE` is available only when the Flow Label support is enabled by `bcmSwitchMplsEvpnEntropyEnable`. The possible values for the Switch Control are as follows:

- 3 – Enable both Flow Label and Flow Label with Control Word (default value).
- 2 – Enable Flow Label with Control Word.
- 1 – Enable Flow Label.
- 0 – Disable the support of Flow Label in EVPN IML.

IML additional attributes:

1. To select FP context as defined in the reference described above, LIF (IML with ESI for the 2 logical-interfaces solution should be set with a user-defined InLIF profile:

```
bcm_gport_t gport;
BCM_L3_ITF_LIF_TO_GPORT_TUNNEL(gport, iml_info.tunnel_id);
bcm_port_class_set(unit, gport, bcmPortClassFieldIngressVport, <profile_value>);
```

2. To configure network group ID (for split horizon filtering)

```
bcm_gport_t gport;
BCM_L3_ITF_LIF_TO_GPORT_TUNNEL(gport, iml_info.tunnel_id);
bcm_port_class_set(unit, gport, bcmPortClassForwardIngress,<network_group_id>);
```

Unknown-DA MACs:

1. To set flooding group offset call `bcm_port_flood_group_set(unit, port, flags, flood_groups)`. Flags are not used currently. For `flood_groups`, it is defined as below:

```
typedef struct bcm_port_flood_group_s {
    bcm_gport_t unknown_unicast_group;
    bcm_gport_t unknown_multicast_group;
    bcm_gport_t broadcast_group;
} bcm_port_flood_group_t;
```

2. Members in the structure are used as an offset to flooding groups for unknown unicast, unknown multicast and broadcast packets. The value used for them is gport that can be:
   - `BCM_GPORT_BLACK_HOLE` – Drop packets on specific InLIF or in-port. See Chapter 2, Ports and Generalized Ports, for additional configuration required for black hole.
   - `BCM_GPORT_TYPE_MCAST` – The Multicast-ID is the offset to flooding groups for Unknown packets.
   - `BCM_GPORT_TYPE_TRAP` – Flooding groups for unknown packets are destined to configured trap.
   - `BCM_GPORT_INVALID` – Keep the flooding group unchanged.

35.14.4 Shell Commands

None

35.14.5 Application Reference

Example application defining IML label termination with and without expected ESI under it

- **Type:** Application reference
- **Path**: `$SDK/src/examples/dnx/cint_dnx_evpn_basic.c`
- **Function**: `iml_term_add`

35.15 E-Tree in EVPN

In an E-Tree service, a customer site that is typically represented by an AC is labeled as either a Root or a Leaf site. It may also be represented by a MAC address along with a VLAN tag. Root sites can communicate with all other customer sites (both Root and Leaf sites). However, Leaf sites can communicate with Root sites only.

In this document, for E-Tree support, a site (Root or Leaf) is represented by an AC, and the MAC address is colored as Root or Leaf in its advertisements.

Thus, when a packet comes from a Leaf site, its destination should be checked first. If the destination is found by an L2 lookup and marked with a Leaf indication, then the packet is dropped. If the destination is not found, a Leaf label should be encapsulated for the copies to the EVPN network. This leaf label is used for filtering the copies to Leaf sites in egress PE. Because the leaf label can also block the traffic to the same ES (Leaf site), no ESI label is needed in this case. If the packet originated from a Root site, only the ESI label is encapsulated according to its destination.

35.15.1 Leaf Label Encapsulation

Because a Leaf label will never be encapsulated together with an ESI label, and it is assigned with a similar mechanism, the ESI encapsulation procedure is recommended for the Leaf label encapsulation. See Section 35.4, ESI Encapsulation.

35.15.2 Leaf Label Termination

The Leaf label is terminated together with IML as ESI. So, the same termination procedure for ESI is also used for Leaf label termination. See Section 35.14, IML Label Termination.

35.15.3 Leaf Traffic Filter by Field Processor

When a packet originating from a Leaf site is destined for another Leaf site, it should be blocked. This is done by the Field Processor.

For a packet from an AC, users can allocate 1 bit or more from the wide-data for the originating Leaf indication. In MACT, the destination Leaf indication should be set in the destination group field (`bcm_l2_addr_t.group`) when the MAC address is advertised by the control plane.

Thus, for unicast traffic, the Field Processor can be configured in the ingress PE to filter the packets from Leaf sites to Leaf sites by qualifying the wide-data using `bcmFieldQualifyAcInLifWideData` and the destination group using `bcmFieldQualifyDstClassL2`. In this way, the filtering needed by an E-Tree service for known unicast traffic is performed very efficiently, and known unicast leaf traffic over the MPLS/IP core is avoided.

For BUM (broadcast, unknown unicast, and unknown multicast) traffic, the filter must be done at the egress PE with Leaf label. Generally, users can design the filter themselves by the Field processor, such as using a filter procedure similar to ESI, or different solutions. The official recommended solution is to define a Leaf-Label-DB in the ingress Field Processor, in which when the Leaf label after IML termination gets matched in the DB, the incoming orientation is updated with a proper value. Then the updated incoming orientation together with the outgoing orientation will filter the packet as expected. Users cannot update the incoming orientation directly by the Field Processor but update the InLIF field information using `bcmFieldActionInVportClass0`.

35.15.4 SOC Properties

None

35.15.5 Shell Commands

None

35.15.6 Application Reference

Example for configuring Leaf indication in inAC for E-Tree

- **Type:** Application reference
- **Path:** `$SDK/src/examples/dnx/cint_dnx_evpn_basic.c`
- **Function:** `evpn_es_add(...)`

Example for configuring Leaf indication in MAC for E-Tree

- **Type:** Application reference
- **Path:** `$SDK/src/examples/dnx/cint_dnx_evpn_basic.c`
- **Function:** `evpn_egress_uc_l2_address_add(...)`

Example for Leaf label encapsulation for E-Tree

- **Type**: Application reference
- **Path**: `$SDK/src/examples/dnx/cint_dnx_evpn_etree.c`
- **Function**: `evpn_etree_leaf_encap_config(...)`

Example for Leaf filter by Field Processor

- **Type:** Application reference
- **Path:** `$SDK/src/examples/dnx/cint_dnx_evpn_etree.c`
- **Function:** `evpn_etree_filter_init(...)`

Example for the simple E-Tree applications in EVPN

- **Type**: Application reference
- **Path**: `$SDK/src/examples/dnx/cint_dnx_evpn_etree.c`
- **Function**: `evpn_etree_main()`

35.16 Recycle in EVPN

Sometimes it is necessary to recycle the packet to the second pass for more PP processing in EVPN applications, such as more MPLS label encapsulations, ECMP on the encapsulated BUM traffic, and so on. Generally, the API sequences for recycling in EVPN are similar to recycling in other applications.

This section includes an example for the outer MPLS label swap in the second pass to explain the usage of recycle in EVPN. For the recycle application work, the configurations should include not only the services in the first pass and second pass, but also the configurations for recycle, such as the recycle port and recycle header. It is important to confirm the recycle port and the related header are configured correctly so that recycled traffic is processed as expected.

For more details about the recycle purposes and their configurations, see Chapter 17, Standardized Recycle Header.

35.16.1 SOC Properties

The following examples set port 40 as the dedicated recycle port:

- `tm_port_header_type_in_40=RCH_0`
- `tm_port_header_type_out_40=ETH`
- `ucode_port_40=RCY.0:core_0.40`

35.16.2 Shell Commands

None

35.16.3 Application Reference

Example for recycle usage in EVPN:

- **Type**: Application reference
- **Path**: `$SDK/src/examples/dnx/cint_dnx_evpn_basic.c`
- **Function**: `evpn_bum_recycle_lsp_swap_set()`

35.17 VXLAN EVPN

The significant differences between VXLAN EVPN and MPLS EVPN are the modifications in the control plane, which are out of the scope of this section. The data plane processing is mostly similar, except for the configurations for packet forwarding and tunnel encapsulation and termination. See Chapter 32, VXLAN IP Overlay for more information about the configurations for VXLAN applications.

One point that deserves mention is the split horizon filter in BUM traffic. Unlike MPLS EVPN, in which the ESI label is used to prevent the traffic looping back to the original ES, the recommended approach to the same purpose referred to as *local bias* in VXLAN encapsulation is as follows.

In multihomed ES, every Network Virtualization Edge (NVE) tracks the IP addresses of other NVEs inside the same ES. When the NVE receives a copy of the BUM packet from the VXLAN network, it examines the source IP address in the VXLAN tunnel header and filters out the packet on all local interfaces connected to the ES that are shared with the ingress NVE. The source IP address actually corresponds to the ingress NVE, thus, indicates the ES. In this approach, the ingress NVE is required to perform the replication locally to all directly attached ESs (regardless of the DF election state) for the BUM traffic from the access interfaces.

Similar to the ESI filter in MPLS EVPN, the egress FP is recommended for the source IP filter with possible key fields of the source IP and out-port that is connected to the shared ES.

35.17.1 SOC Properties

None

35.17.2 Shell Commands

None

35.17.3 Application Reference

Example for MPLS EVPN from/to VXLAN EVPN:

- **Type**: Application reference
- **Path**: `$SDK/src/examples/dnx/cint_dnx_evpn_vxlan.c`
- **Function**: `evpn_vxlan_main(...)`

35.18 API Descriptions

35.18.1 bcm_mpls_tunnel_initiator_*

API Name	Highlights
`bcm_mpls_tunnel_initiator_create()`	Can be used to create EVPN/IML encapsulation
`bcm_mpls_tunnel_initiator_get()`	Can be used to retrieve EVPN/IML encapsulation details
`bcm_mpls_tunnel_initiator_clear()`	Can be used to delete EVPN/IML encapsulation entry
`bcm_mpls_tunnel_initiator_traverse()`	Can be used to traverse EVPN/IML/IML+ESI label encapsulations.

35.18.2 bcm_mpls_esi_encap_*

API Name	Highlights
`bcm_mpls_esi_encap_add()`	Add ESI encapsulation entry in ESEM dual homing DB
`bcm_mpls_esi_encap_get()`	Retrieve ESI encapsulation entry from ESEM dual homing DB
`bcm_mpls_esi_encap_delete()`	Delete ESI encapsulation entry from ESEM dual homing DB
`bcm_mpls_esi_encap_traverse()`	Call a user-defined callback over all the ESI entries in ESEM dual homing DB.

35.18.3 bcm_mpls_tunnel_switch_*

API Name	Highlights
`bcm_mpls_tunnel_switch_create()`	Can be used to create EVPN/IML termination
`bcm_mpls_tunnel_switch_get()`	Can be used to retrieve EVPN/IML encapsulation details
`bcm_mpls_tunnel_switch_delete()`	Can be used to delete EVPN/IML encapsulation
`bcm_mpls_tunnel_switch_traverse()`	Can be used to traverse EVPN/IML/IML+ESI label terminations.

35.18.4 bcm_mpls_range_action_*

API Name	Highlights
`bcm_mpls_range_action_add()`	Create an action range of MPLS labels. This should be used for specifying a range in which labels are considered IML for termination with/without ESI.
`bcm_mpls_range_action_get()`	Get the action of a range of MPLS labels
`bcm_mpls_range_action_remove()`	Deletes an action range of MPLS labels

Chapter 36: IEEE 1588v2 Precision Time Protocol

36.1 Introduction

IEEE1588v2 Precision Time Protocol (PTP) is a distributed synchronization protocol that specifies how the real-time clocks in the system synchronize with each other.

These clocks are organized into a timeTransmitter/timeReceiver synchronization hierarchy, with the clock at the top of the hierarchy, the Grandmaster clock, determining the reference time for the entire system. Synchronization is achieved by exchanging PTP timing messages, with the timeReceivers using the timing information to adjust their clocks to the time of their timeTransmitters in the hierarchy. Typically, within a system (for example, switch) there is one device that synchronizes the clock and distributes it to the devices within the system by using a BroadSync® interface.

36.1.1 Hardware Components

The IEEE 1588 feature involves the following hardware components:

- Host processor – A general-purpose CPU shared with the customer application.
- Timing processor (TOP) – A limited-purpose CPU specific to 1588 application acceleration or tight coupling with hardware. In the DNX device it is the CMICx microcontroller.
- Timing Block – A dedicated hardware block within the device (packet processing pipe and PCS) for time stamping and time synchronization. Synchronization is system-wide.
- PHY – Physical layer discrete device. Capable of one-step time stamping.

36.1.2 Software Components

The IEEE 1588 feature involves the following software components:

- SDK – Broadcom common API library, which resides in the host. It configures the device's timing block, and includes PTP APIs and the TOP firmware image (for downloading).
- BroadSync software – Firmware that poles the external low-frequency clock from the OC and synchronizes local timers.
- Servo/1588v2 stack – Firmware to offload 1588 functionality from the host processor.

36.1.3 IEEE1588v2 Supported Modes

The device design supports IEEE1588v2 protocol in one of the following modes:

- End-to-end transparent clock, one-step or two-step. For two-step end-to-end transparent clock, additional software support is needed.
- Peer-to-peer transparent clock, one-step or two-step. For peer-to-peer transparent clock, additional software is needed to run on host CPU.
- Boundary clock, one-step or two-step. Broadcom BCM 1588 firmware that runs on the internal processor (iProc) dual ARM co-processor may be used for the boundary clock.
- Ordinary clock, one-step or two-step. Broadcom BCM 1588 firmware that runs on the internal processor (iProc) dual ARM co-processor may be used for the ordinary clock.

36.1.4 IEEE 1588v2 (PTP) Encapsulation

The device supports various types of encapsulations.

For example:

- IEEE-1588 over Ethernet
- IEEE-1588 over UDP over IP
- Flexible encapsulations over IP/MPLS tunnel
  - UDP encapsulation over IP tunnel – 1588oUDPoIPoIPoETH.
  - UDP encapsulation over MPLS tunnel – 1588oUDPoIPoMPLSoMPLSoETH.
  - ETH encapsulation over MPLS tunnel – 1588oETHoMPLSoMPLSoETH.

NOTE: IP includes both IPv4 and IPv6.

36.1.5 IEEE1588v2 Packet Processing

This section describes the functionality within the packet processing pipe for processing I588v2 (ptp) packets.

One-Step E2E Transparent Clock Packet Walk

In the most basic architecture, the system acts as a transparent end-to-end device. As a PTP packet enters the device, it goes through the following steps:

1. The ingress PCS provides an updated 32-bit time-stamp to the packet.
2. The Ingress Receive Packet Processor (IRPP) identifies the PTP packet and marks it as such. The IRPP appends to the packet information on its nature (one-step mode, PTP header offset within the packet, and so on).
3. The label processor creates the application-specific extension header with the information processed by the IRPP and timestamp header (TSH) that carry the RX PCS timestamp. The information continues with the packet through the pipe.

NOTE: If system Header `mode=jericho`, the information is carried by the OAM-TS system header.

4. The packet traverses the DNX system. It may be locally routed in the same device, or go through the Fabric.
5. As the packet arrives at Egress Transmit Packet Processor (ETPP), the application specific extension header is processed. The ETPP conducts the following actions on the packet:
   a. Update the correction field (CF) value based on the receive time and egress port delay configuration.
   b. Update the UDP checksum, if the packet encapsulation is UDP.
   c. Generate control signals to the Egress PCS, indicating PTP packet and header offset.
6. The Tx PCS stamps the packet's CF field with the transmit time and updates UDP checksum if needed.
7. The packet is sent to its destination.

Two-Step E2E Transparent Clock Packet Walk

A two-step end-to-end transparent clock has the same system-level physical implementation as a one-step end-to-end transparent clock. The main difference between the two implementations is the way the packets are handled by the device and the requirements from the system’s management.

As a PTP packet enters the device with two-step flag, it goes through the following steps:

1. The ingress PCS provides an updated 32-bit time-stamp to the packet.
2. The Ingress Receive Packet Processor(IRPP) identifies the PTP packet and marks it as such. The IRPP appends to the packet information on its nature (two-step mode, PTP header offset within the packet, and so on), and in addition marks that the packet needs to be trapped and sent to the clock module.
3. The label processor creates the application specific extension header with the information processed by the IRPP and timestamp header (TSH) which carry the RX PCS timestamp. The information continues with the packet through the pipe.

NOTE: If system header mode=jericho, the information is carried by the OAM-TS system header.

4. The packet traverses the DNX system. It may be locally routed in the same device, or go through the Fabric.
5. As the packet arrives at Egress Transmit Packet Processor (ETPP), the ETPP generates the command for Tx PCS to perform two-step time stamping(store the packet transmit time into a fifo). (See details in the next section, Follow-up message generation.)
6. The packet is sent to its destination.

Follow-up Message Generation

1. The trapped packet is sent to the clock module through a CPU port.
2. The clock module processes the arriving packet:
   a. Extracts the the 32-bit arrival time-stamp and 1588 encapsulation offset.
   b. Accesses the device and reads from Tx PCS FIFO the original packet transmit time.
   c. Calculates the residence time for the follow-up message.
3. The clock module generates a new PTP packet (for example, Follow_Up message).
4. The IRPP identifies the PTP packet and marks it as such.
5. The packet traverses the DNX system locally.
6. As the packet arrives at the Egress, there is no need to perform any actions on the message neither by the ETPP nor by the Tx PCS.
7. The packet is sent to its destination.

P2P Transparent Clock

A Peer-to-Peer transparent clock DNX system operates very much like the end-to-end transparent clock (see the packet walk information above). The Link delay is configured in the ingress port-delay register, thus if the device is configured to compensate for the port delay, the delay is added to the correction field.

36.1.6 Application Configuration Checklist

- PTP general setup
- TRAP configuration
- PTP Port Configuration

36.1.7 PTP General Setup

General configuration is required to make the PTP system work.

36.1.8 SOC Properties

1. To map a local CPU port (for example: 204) to ARM queue:
   ```
   num_queues_pci=40
   num_queues_uc1=8
   ucode_port_204=CPU.40:core_0.204
   ```
2. To tell host CPU that local port 204 was mapped to cpu ARM (Used to encapsulate PTCH when sending packets from Firmware):
   ```
   custom_feature_ptp_cosq_port=204
   ```
3. To trap 1588 packets to uKernel (ARM) without dune header (This is the ARM RX port):
   ```
   tm_port_header_type_out_204=ETH
   ```
4. To send ARM TX packets into the device, carry special header (such as PTCH):
   ```
   tm_port_header_type_in_0=INJECTED_2
   ```
5. To enable local TS_PLL clock:
   ```
   phy_1588_dpll_frequency_lock=1
   ```
6. To enable brodSync clock:
   ```
   BroadSync_enable_clk=1
   ```

36.2 Trap Configuration

The `bcm_rx_trap_type_create` trap API with `type=bcmRxTrap1588/bcmRxTrap1588User1-5` and `bcm_rx_trap_set` configure the ingress trap action for IEEE 1588 packets.

A usage example is to trap general PTP messages into the CPU.

For the API configuration sequence, see Chapter 12, Traps.

36.2.1 Application Reference

IEEE 1588 Trap Configuration

Example of 1588 PTP configuration.

- **Type:** CINT reference
- **Path:** `cint_ptp_1588.c`
- **Function**: `ieee_1588_trap()`

IEEE 1588 User Trap Configuration

Example of 1588 PTP user trap configuration.

- **Type**: CINT reference
- **Path**: `cint_rx_trap_1588_traps.c`

36.3 PTP Port Configuration

To configure the PTP configuration per port, use the `bcm_port_timesync_config_set` API.

To configure the PTP configuration in PCS, use the `bcm_port_phy_timesync_config_set` API.

Use the `bcm_port_control_set` API (`type=bcmPortControl1588P2Pdelay`) to configure a const P2P delay (nanoseconds) per port. Note that a negative value is not accepted when `type=bcmPortControl1588P2Pdelay`.

If Linkscan is on, use the API `bcm_port_phy_timesync_config_set` to configure the PTP configuration in PCS. This API does not enable RX timestamping directly. RX timestamping is enabled by Linkscan on every link up event and disabled on every link down event.

If Linkscan is off, use the API `bcm_port_control_phy_timesync_set` to enable all PTP functionality in PCS on every link up event and disable all PTP functionality in PCS on every link down event.

36.3.1 Configuration Flow

1. Set SOC properties according to the desired ports used for the PTP protocol. See Section 36.1.8, SOC Properties.
2. Configure an ingress trap for PTP packets if needed using following APIs:

   `bcm_rx_trap_type_create(unit, flags, trap_type, &trap_id), type=bcmRxTrap1588/ bcmRxTrap1588User1-5`

   `bcm_rx_trap_set(unit, trap_id, &trap_config)`
3. Configure the relevant ptp parameters per port, using the API `bcm_port_timesync_config_set(unit, port, config_count, config_array)`
   – `flags` – The flags in the following table are supported.

Flag Name	Explanation
`BCM_PORT_TIMESYNC_DEFAULT`	Always on
`BCM_PORT_TIMESYNC_ONE_STEP_TIMESTAMP`	If set, 1588 messages Sync, Delay_req, Pdelay_req, and Pdelay_resp are recorded in Ingress and the CF is stamped in TX.
`BCM_PORT_TIMESYNC_TWO_STEP_TIMESTAMP`	If set, the 1588 messages Sync, Delay_req, Pdelay_req, and Pdelay_resp time are recorded in ingress. At the egress, the transmit time is recorded in a FIFO and is read by the 1588 controller.
`BCM_PORT_TIMESYNC_EXTERNAL_MAC_ENABLE`	if set, defines to use external mac for that port.

   – `pkt_drop, pkt_tocpu`:
   Bitmaps of 1588 commands. A 1588 packet command that appears in pkt_tocpu is trapped to the 1588 host, according to trap code bcmRxTrap1588 configuration or one of bcmRxTrap1588User1-5 trap codes given as `user_trap_id`. A 1588 packet command that appears only in `pkt_drop` is dropped. A packet that does not appear in either `pkt_tocpu` and `pkt_drop` is forwarded transparently.

   The command types are:

   - `BCM_PORT_TIMESYNC_PKT_SYNCH`
   - `BCM_PORT_TIMESYNC_PKT_DELAY_REQ`
   - `BCM_PORT_TIMESYNC_PKT_PDELAY_REQ`
   - `BCM_PORT_TIMESYNC_PKT_PDELAY_RESP`
   - `BCM_PORT_TIMESYNC_PKT_FOLLOWUP`

- BCM_PORT_TIMESYNC_PKT_DELAY_RESP
- BCM_PORT_TIMESYNC_PKT_PDELAY_RESP_FOLLOWUP
- BCM_PORT_TIMESYNC_PKT_ANNOUNCE
- BCM_PORT_TIMESYNC_PKT_SIGNALLING
- BCM_PORT_TIMESYNC_PKT_MANAGMENT
- BCM_PORT_TIMESYNC_PKT_INVALID

– user_trap_id:
To trap the IEEE 1588 packet according to one of bcmRxTrap1588User1-5 traps, supply the trap_id of the trap configured as user_trap_id.

4. If Linkscan is on, configure the relevant 1588 feature in PCS, using the API
bcm_port_phy_timesync_config_set(unit, port, timesynce_config)
– flags – The flags in the following table are supported.

Flag Name	Explanation
BCM_PORT_PHY_TIMESYNC_ENABLE	Enable 1588 feature in PCS.
BCM_PORT_PHY_TIMESYNC_ONE_STEP_ENABLE	Enable 1588 one-step timestamping. If this flag is not set, two-step timestamping is enabled.

NOTE:
- PCS RX timestamping is not enabled by this API. Linkscan enables RX timestamping on every Link up event.
- Ports need to be disabled before calling this API.

5. If Linkscan is off, configure the relevant 1588 feature in PCS, using the API
bcm_port_control_phy_timesync_set(unit, port, type, val):
– flags – The flags in the following table are supported

Flag Name	Explanation
bcmPortControlPhyTimesyncTimestampAdjust	Timestamp adjustment mode.
bcmPortControlPhyTimesyncOneStepEnable	Enable 1588 one-step timestamping.

6. Configure a constant value for ptp delay if needed, using the API bcm_port_control_set (type=bcmPortControl1588P2Pdelay).

36.3.2 Application Reference

One-step or two-step PTP configuration

Example of IEEE 1588 PTP configuration.
- **Type:** CINT reference
- **Path:** cint_ptp_1588.c

36.4 API Descriptions

36.4.1 bcm_port_timesync_config_*

API Name	Highlights
`bcm_port_timesync_config_set()`	Set PTP pipe configuration per port
`bcm_port_timesync_config_get()`	Get PTP pipe configuration per port

36.4.2 bcm_port_phy_timesync_config_*

API Name	Highlights
`bcm_port_phy_timesync_config_set()`	Set 1588 configuration in PCS.
`bcm_port_phy_timesync_config_get()`	Get 1588 configuration in PCS.

36.4.3 bcm_port_control_phy_timesync_*

API Name	Highlights
`bcm_port_control_phy_timesync_set()`	Enable PCS Rx timestamping if Linkscan is off.
`bcm_port_control_phy_timesync_get()`	Get PCS Rx timestamping enable state.

Chapter 37: ITU-T G.8032 Ring Automatic Protection Switching

37.1 Introduction

An Ethernet automatic protection switching (APS) ring has a physical ring topology. Each ring is made up of ring protection links (RPLs) between the participating nodes.

The Ethernet ring is attached to the switch on two ports, referred to as the Left-Port and Right-Port. The switch learns that some of the devices on the Ethernet ring protection (ERP) are on the Left-Port and some are on the Right-Port, reflecting the way the ring was broken.

When a failure is identified on an RPL, the following actions occur:

- The RPL that was blocked is reinstated.
- The ring node adjacent to the RPL blocks traffic to the ring and from it.

37.2 Application Configuration Checklist

- AC-LIF creation with a flush group association
- MAC table flush per group
- ERP block and unblock

37.3 AC-LIF Creation with a Flush Group Association

An AC-LIF associated with a ring protection scheme is created like any AC-LIF (see Chapter 21, Ethernet Bridge).

37.3.1 SOC Properties

None

37.3.2 Configuration Flow

Create a VLAN-Port by calling `bcm_vlan_port_create(unit, vlan_port)`:

- `vlan_port.group` – A flush group that is unique per ring port
  For ERP blocking, see Section 37.5, Block and Unblock ERP.
  For other fields, see Section 27.4, Creating Service AC-LIFs.

It is possible to move a LIF from one ring port to another by calling the creation API with the `BCM_VLAN_PORT_REPLACE` and `BCM_VLAN_PORT_WITH_ID` flags. The group ID and the ERP blocking fields should be modified, while other AC-LIF parameters should remain with the same values. However, modifying a standard AC-LIF to an ERP AC-LIF (and vice versa) is not allowed.

A limitation is that setting the group field is invalid for an optimized VLAN-Translation AC (`vlan_port.flags` set to `BCM_VLAN_PORT_VSI_BASE_VID`), and setting the group field also disables ingress wide data of more than 8 bits (using the `bcm_port_wide_data_set()` API).

37.3.3 Shell Commands

None

37.3.4 Application Reference

`$SDK/src/examples/dnx/cint_ring_protection.c`

37.4 MAC Table Flush Per Group

Perform a MAC table flush only for entries that are associated with a specific flush group that represents one of the ring ports in a ring topology.

The user is expected to perform a flush for all the ring ports on all nodes participating in a ring that has experienced a state change of one of its RPLs.

37.4.1 SOC Properties

None

37.4.2 Configuration Flow

Perform a MAC table flush by calling `bcm_l2_replace(unit, flags, match_addr, new_module, new_port, new_trunk)`:

- `flags` – The following flags are required to achieve a flush of all possibly related entries:
  - `BCM_L2_REPLACE_DELETE`
  - `BCM_L2_REPLACE_NO_CALLBACKS`
  - `BCM_L2_REPLACE_IGNORE_DES_HIT`
  - `BCM_L2_REPLACE_MATCH_STATIC`
  - `BCM_L2_REPLACE_MATCH_GROUP`
- `match_addr.group` – The flush group ID associated with the ring port.

NOTE: A flush performed with these flags will also remove static entries that were created with the `bcm_l2_addr_add()` API using a valid `l2_addr.group` field.

37.4.3 Shell Commands

None

37.4.4 Application Reference

`$SDK/src/examples/dnx/cint_ring_protection.c`

37.5 Block and Unblock ERP

An ERP node adjacent to an RPL that either fails or recovers is required to respectively block or unblock traffic from the ring port connected to the stated RPL.

The block and unblock operations should affect both the incoming traffic to the ring port as well as the outgoing traffic from it.

The block operation should be performed prior to the MAC table flush operation, while an unblock should be performed after the MAC table flush operation.

37.5.1 SOC Properties

None

37.5.2 Configuration Flow

Create an ERP RX blocking ingress protection object by calling `bcm_failover_create(unit, flags, *failover_id)`:

- `flags` – `BCM_FAILOVER_INGRESS`
- `failover_id` – The returned failover ID that represents the Rx-Blocking of a specific Ring-Port.

Create an ERP Tx-Blocking egress protection object by calling `bcm_failover_create(unit, flags, *failover_id)`:

- `flags` – `BCM_FAILOVER_ENCAP`
- `failover_id` – The returned failover ID that represents the Tx-Blocking of a specific Ring-Port.

To perform both TX and RX ERP blocking and unblocking for a ring port, perform the following steps:

1. Set the ERP Rx-Blocking state by calling `bcm_failover_set(unit, failover_id, enable)`.
   - `failover_id` – The failover ID that represents the Rx-Blocking of a specific ring port.
   - `enable`:
     - Use 1 to block the traffic.
     - Use 0 to unblock the traffic.
2. Set the ERP TX-blocking state by calling `bcm_failover_set(unit, failover_id, enable)`.
   - `failover_id` – The failover ID that represents the TX-blocking of a specific ring port.
   - `enable`:
     - Use 1 to block the traffic.
     - Use 0 to unblock the traffic.

37.5.3 Shell Commands

None

37.5.4 Application Reference

`$SDK/src/examples/dnx/cint_ring_protection.c`

37.6 API Descriptions

API Name	Highlights
`bcm_vlan_port_create()`	Create an ERP AC-LIF or modify it.
`bcm_vlan_port_find()`	Retrieve ERP parameters of an AC-LIF.
`bcm_l2_replace()`	Perform MAC table flush operation per a Ring-Port.
`bcm_failover_create()`	Create an Ingress or Egress failover object associated with a Ring-Port.
`bcm_failover_destroy()`	Destroy either an Ingress or Egress failover object associated with a Ring-Port.
`bcm_failover_set()`	Block or Unblock traffic through an Ingress or Egress failover object associated with a Ring-Port.
`bcm_failover_get()`	Retrieve Block or Unblock state for a failover object associated with a Ring-Port.

Chapter 38: Mirror Protocols

38.1 Introduction

Mirroring is one of the applications supported by SNIFF in the BCM88690. For more information about Sniff, refer to the Sniff chapter in the *Traffic Manager Programming Guide*. As part of the Sniff applications, packet-processing is responsible for the following aspects:

- Trigger Sniff action, which is done by the Mirror-Snoop profile pointed to by <port, vlan> pair at ingress (inbound mirroring) or egress (outbound mirroring), by the flexible field processor, or by the trap code mechanism.
- Generate or overwrite (or generate and overwrite) Sniff copy system-headers for various mirroring-protocols (for example, SPAN, RSPAN, ERSPAN, Lawful-Interception).

38.2 Application Configuration Checklist

1. Create a Mirror-profile. This is done by calling `bcm_mirror_destination_create`, refer to the Sniff chapter in the *Traffic Manager Programming Guide*.
2. Create the mirroring application: RSPAN, RSPAN advanced, ERSPANv2, ERSPANv3, or Lawful-Interception tunnels.
3. Configure system-headers of the Mirror-profile (see Section 38.3, Mirror Header Generation).
4. Trigger Mirror-profile to a packet by using <port, vlan> pair (`bcm_mirror_port_vlan_destination_add`), Field Processor, or the trap-code mechanism

38.3 Mirror Header Generation

For a mirrored or snooped copy, the BCM88690 device can append a new system header at the beginning of the packet (append mode), or overwrite the original system header with new values (overwrite mode), before sending the packet to the fabric.

38.3.1 SOC Properties

None

38.3.2 Configuration Flow

To create a snoop or mirror profile, call `bcm_mirror_destination_create(int unit, bcm_mirror_destination_t * mirror_dest);`

- API fields are fully explained in the Sniff chapter of the *Traffic Manager Programming Guide*.
- For outbound snoop packets to the CPU, it is possible to carry original system headers by setting `BCM_MIRROR_DEST_EGRESS_ADD_ORIG_SYSTEM_HEADER`.
- For inbound snoop packets to CPU, it is possible to overwrite existing system headers and set the snoop or trap qualifier in the fabric header by calling `BCM_MIRROR_PKT_HEADER_UPDATE_FABRIC_HEADER_EDITING` flag. That is, `mirror_dest->snoop_dest.packet_control_updates.valid = BCM_MIRROR_PKT_HEADER_UPDATE_FABRIC_HEADER_EDITING`

To set new system headers for a snoop or mirror profile, call `bcm_mirror_header_info_set(int unit, uint32 flags, bcm_gport_t mirror_dest_id, bcm_mirror_header_info_t *mirror_header_info)`

- When the API is called, append mode (new system-headers) is enabled. This is the recommended way to work in JR2 system-headers mode.
- The same mirror profile needs to be destroyed before it can be used to work in non-append mode again.

NOTE: The new system-headers will be added on top of existing system-headers or overwritten according to the `BCM_MIRROR_DEST_EGRESS_ADD_ORIG_SYSTEM_HEADER` flag.

- `flags`

Flags	Description
`BCM_MIRROR_DEST_IS_SNOOP`	`mirror_dest_id` is of snoop type (default is mirror)
`BCM_MIRROR_DEST_IS_STAT_SAMPLE`	`mirror_dest_id` is of statistic sampling type (default is mirror)
`BCM_MIRROR_DEST_EGRESS_ADD_ORIG_SYSTEM_HEADER`	If asserted, a new system header will be added on top of existing system-headers; otherwise, the new system header overwrites existing system-headers.
`BCM_MIRROR_DEST_STAMP_IVE_ACTION_ID`	Set the IVE action ID to mirror the system header if asserted.

- `mirror_dest_id` – Mirror destination ID, returned by `bcm_mirror_destination_create`.
- `mirror_header_info.tm.src_sysport` – Source system port ID to be stamped for the SNIFF copy.
- `mirror_header_info.tm.tc` – Internal traffic class for the SNIFF copy.
- `mirror_header_info.tm.dp` – Internal drop precedence for the SNIFF copy (currently reserved).
- `mirror_header_info.tm.is_mc_traffic` – Indicates whether the SNIFF copy should be replicated at egress. If set, a multicast ID will be stamped in the FTMH header, otherwise, an OutLIF will be stamped in the FTMH header.
- `mirror_header_info.tm.mc_id` – When `is_mc_traffic` is set, gives the multicast ID to be stamped for the SNIFF copy.

- `mirror_header_info.tm.out_vport` – When `is_mc_traffic` is cleared, this is the gport of the OutLIF to be stamped for the SNIFF copy.
- `mirror_header_info.tm.ase_ext.valid` – Whether the SNIFF copy should have an application specific extension header.
- `mirror_header_info.tm.ase_ext.ase_type` – Type of the application specific extension header, refer to `bcm_pkt_dnx_ase_type_t` for all supported values.

To support remote CPU processing where the fabric system-headers of the original forwarding copy must be part of the outgoing packet, and a remote CPU encapsulation header is required, call `bcm_mirror_header_info_set`, which defines the remote CPU encapsulation, and set the `BCM_MIRROR_DEST_EGRESS_ADD_ORIG_SYSTEM_HEADER` flag.

The fabric system-headers of the original forwarding copy will be kept from the ingress PP to the egress PP and will be built as part of the payload and sent to the remote CPU. If using the `ENCAP_EXTERNAL_CPU` port header type for remotely connected CPU processing, do not call `bcm_mirror_header_info_set`. This case requires the egress PMF to overwrite the forwarding-additional-information (FAI) to be bridging because the FAI carries the information of the original forwarding copy from ingress to egress.

38.3.3 Application Reference

Example of mirror protocols.

- **Type:** CINT reference
- **Path:** `$SDK/src/examples/dnx/cint_dnx_mirror_example.c`

38.4 RSPAN Tunnel

An RSPAN tunnel is a VLAN tunnel dedicated to SPAN copied traffic. Packets mirrored to a RSPAN tunnel have an additional VLAN tag.

38.4.1 SOC Properties

None

38.4.2 Configuration Flow

A RSPAN tunnel can be built by creating an egress only AC lif, with a VLAN translation action that adds an additional RSPAN tunnel tag for all known tag formats of packets.

1. Create an egress AC LIF by calling `bcm_vlan_port_create (unit, bcm_vlan_port_t * vlan_port);`
   - `vlan_port`
     - `flags = BCM_VLAN_PORT_CREATE_EGRESS_ONLY`
     - `criteria = BCM_VLAN_PORT_MATCH_NONE`

     For more information about creating AC LIF, see Chapter 21, Ethernet Bridge.
2. Configure VLAN translation for the AC lif by doing the following:
   a. Create a VLAN translation action that adds an additional tag to the packet (if such a VLAN translation action does not exist already) by calling b`cm_vlan_translate_action_id_create (unit, flags, action_id);`
      - `flags = BCM_VLAN_ACTION_SET_EGRESS`
      - `action_id` – the returned action_id allocated by SDK

b. Set action for the created VLAN translation action ID by calling `bcm_vlan_translate_action_id_set (unit, flags, action_id, action);`
   - `flags = BCM_VLAN_ACTION_SET_EGRESS`
   - `action_id` – the action ID created above
   - `action`
   - `outer_tpid` – TPID of RSPAN tag

     `dt_outer = bcmVlanActionAdd`

     `dt_outer_pkt_prio can be one of the following:`
     - `bcmVlanActionOuterAdd` – RSPAN PCP-DEI is taken from outer tag
     - `bcmVlanActionInnerAdd` – RSPAN PCP-DEI is taken from inner tag
     - `bcmVlanActionReplace` – RSPAN PCP-DEI is taken from AC LIF

c. Reserve a VLAN editing profile to add an additional VLAN tag for all known VLAN tag formats (if such a VLAN editing profile does not already exist) by calling `bcm_vlan_translate_action_class_set(unit, action_class);`

   `action_class`
   - `vlan_edit_class_id` – The VLAN editing profile to be reserved for RSPAN (adding one additional tag for all tag formats)
   - `vlan_translation_action_id` – The VLAN translation action ID created above
   - `flags = BCM_VLAN_ACTION_SET_EGRESS`
   - `tag_format_class_id` – Iterate through all defined tag formats in the system, call `bcm_vlan_translate_action_class_set` for each of them

d. Associate the VLAN editing profile to AC lif and set the RSPAN tag by calling `bcm_vlan_port_translation_set(unit, vlan_port_translation);`

   `vlan_port_translation`
   - `gport` – the VLAN port created in step 1
   - `vlan_edit_class_id` – The VLAN editing profile reserved above
   - `flags = BCM_VLAN_ACTION_SET_EGRESS`
   - `new_outer_vlan` – The RSPAN tag

     For more details about AC-LIF, see Chapter 21, Ethernet Bridge.

     For more details about VLAN editing, see Chapter 16, VLAN Editing Mechanism.

3. Create a mirror profile and point to the RSPAN tunnel created in Step 1 by calling `bcm_mirror_destination_create(unit, mirror_dest);`
   - `mirror_dest`
     - `gport` – mirror destination port gport

4. Set mirror profile header editing information by calling `bcm_mirror_header_info_set(unit, flags, mirror_dest_id, mirror_header_info);`
   - `mirror_dest_id` – Mirror destination ID (`mirror_dest->mirror_dest_id` returned in Step 3).
   - `mirror_header_info`
     - `ftmh.src_sysport = in_port`
     - `ftmh.out_vport` – Point to RSPAN AC-LIF (the `vlan_port_id` returned in Step 1).

   For more details of the parameters for `bcm_mirror_destination_create`, refer to the Sniff chapter in the *Traffic Manager Programming Guide*.

38.4.3 Application Reference

Example of RSPAN configuration sequence:

- **Type:** CINT reference
- **Path:** `$SDK/src/examples/dnx/cint_dnx_mirror_example.c:mirror_rspan_example`

38.5 Advanced RSPAN Tunnel

In advanced mode, an RSPAN tunnel can be defined per monitored port. This is done by creating an RSPAN advanced entry in the egress encapsulation database, and pointing to it from the mirror profile. The RSPAN advanced entry results in an AC lookup using RSPAN monitored port as part of the key. The result is an AC entry which has the necessary VLAN editing information to build the RSPAN tunnel.

38.5.1 SOC Properties

None

38.5.2 Configuration Flow

1. Create a RSPAN Advanced entry in the egress encapsulation database by calling `bcm_vlan_port_create (unit, bcm_vlan_port_t * vlan_port);`

   `vlan_port`
   - `flags = BCM_VLAN_PORT_CREATE_EGRESS_ONLY | BCM_VLAN_PORT_VLAN_TRANSLATION`
   - `criteria = BCM_VLAN_PORT_MATCH_NAMESPACE_PORT`
   - `port` – Source system port or source trunk group of the mirrored packet
   - `match_class_id` – Namespace of the source system port
2. To define VLAN translation action for RSPAN Advanced tunnel, follow Step 2 in Section 38.4.2, Configuration Flow, for RSPAN tunnels.
3. To create a mirror profile and point to the Advanced RSPAN tunnel entry, follow Step 3 in Section 38.4.2, Configuration Flow, for RSPAN tunnels.

38.5.3 Application Reference

Example of Advanced RSPAN configuration sequence.

- **Type:** CINT reference
- **Path:** `$SDK/src/examples/dnx/cint_dnx_mirror_example.c:mirror_rspan_advanced_example`

38.6 ERSPANv2 Tunnel

ERSPAN supports remote monitoring over multiple switches across the network using IPv4 tunnels. The mirrored traffic is switched over the IPv4 tunnels and dedicated ERSPAN tunnel with GRE to the RSPAN session destination.

The ERSPANv2 header contains the following fields:

- Version – 0x1 (ERSPAN II)
- VLAN – ERSPAN tag
- CoS

- En – Trunk encapsulation type associated with the ERSPAN source port for ingress ERSPAN traffic. Possible values are:
  - 00 – Originally without VLAN tag
  - 01 – Originally ISL encapsulated
  - 10 – Originally 802.1Q encapsulated
  - 11 – VLAN tag preserved in frame
- T – Truncated indication
- Session ID – ERSPAN session ID
- Reserved – All bits are set to 0
- Index[19:0] – 20 bit index/port number = {Index[19:17], 1'bDirection, 16'bPort}

38.6.1 SOC Properties

None

38.6.2 Configuration Flow

1. Create an IPv4/IPv6 GRE tunnel by calling `bcm_tunnel_initiator_create (unit, intf, tunnel);`
   - `tunnel`
     - `type = bcmTunnelTypeGreAnyIn4/bcmTunnelTypeGreAnyIn6`
     - `flags = BCM_TUNNEL_INIT_GRE_WITH_SN`
     - `dip/dip6` – DIP of the IPv4/IPv6 tunnel
     - `sip/sip6` – SIP of the IPv4/IPv6 tunnel
     - `ttl` – TTL of the IP tunnel
     - `dscp` – DSCP of the IP tunnel
     - `egress_qos_model.egress_qos = bcmQosEgressModelPipeNextNameSpace`
     - `encap_access = bcmEncapAccessTunnel2`

   For more details of configuring a IPv4 tunnel, see Chapter 29, IP Tunnel v4 Encapsulation.
2. Create an ERSPANv2 tunnel by calling `bcm_tunnel_initiator_create (unit, intf, tunnel);`
   - `tunnel`
     - `type = bcmTunnelTypeErspan`
     - Optionally, set `flags = BCM_TUNNEL_INIT_ERSPAN_WITH_SN` to enable sequence number counting in the GRE header. If sequence number counting is required, the counter engine must be initialized for it. (For more details, see `erspan_init_counter_engine()` in `cint_dnx_mirror_example.c`).
     - `l3_intf_id` – Interface ID of IPv4 GRE tunnel created above (returned in the intf l3a_intf_id in Step 1.
     - `encap_access = bcmEncapAccessTunnel1`
     - `aux_data = Index bits [19:17] of ERSPANv2 tunnel`
3. Create a mirror profile by calling `bcm_mirror_destination_create(unit, mirror_dest)`. A detailed parameter definition of this API is described in the section *Sniff* in the *Traffic Manager Programming Guide*.
4. Set mirror profile header editing information by calling `bcm_mirror_header_info_set(unit, flags, mirror_dest_id, mirror_header_info);`
   - `mirror_dest_id` – mirror destination ID (mirror_dest->mirror_dest_id returned in step 3 above).
   - `mirror_header_info`
     - `tm.src_sysport = in_port`
     - `tm.ase_ext.valid = TRUE`

- `tm.ase_ext.ase_type = bcmPktDnxAseTypeErspan`
- `tm.ase_ext. ase_info.erspan_info.direction` – Direction field in ERSPANv2 header (bit 16 of “Index”).
- `tm.ase_ext. ase_info.erspan_info.truncated_flag` – Truncated indication field in ERSPANv2 header.
- `tm.ase_ext. ase_info.erspan_info.en` – Trunk encapsulation type field in ERSPANv2 header
- `tm.ase_ext. ase_info.erspan_info.cos` – CoS field in ERSPANv2 header
- `tm.ase_ext. ase_info.erspan_info.vlan` – VLAN field in ERSPANv2 header
- `tm.out_vport` – point to ERSPANv2 tunnel (tunnel is the `tunnel_id` returned in *step 2*)

5. `ERSPAN.Session ID` is the ID provided from the mirror qualifier. The mirror qualifier can be updated using ingress Field-Processor.

38.6.3 Application Reference

Example of ERSPANv2 configuration sequence.

- **Type:** CINT reference
- **Path:** `$SDK/src/examples/dnx/cint_dnx_mirror_example.c:mirror_erspan_example_v2`

Example of configuring a mirror qualifier using ingress PMF.

- **Type**: CINT reference
- **Path**: `$SDK/src/examples/dnx/field/cint_field_mirror.c:cint_field_mirror_main`

38.7 ERSPANv3 Tunnel

ERSPANv3 tunnel is similar to ERSPANv2 tunnel, but differs in the ERSPAN header.

The ERSPANv3 header contains the following fields:

- Version – 0x2 (ERSPAN III)
- VLAN – ERSPAN tag
- CoS – Class of Service of the monitored frame
- BSO (Bad/Short/Oversized) – A 2-bit value indicating the integrity of the payload carried by ERSPAN:
  - 00 – Good frame with no error, or unknown integrity
  - 11 – Payload is a bad frame with CRC or alignment error
  - 01 – Payload is a short frame
  - 10 – Payload is an oversized frame
- T (Truncated) – This bit indicates that the frame copy encapsulated in the ERSPAN packet has been truncated
- Session ID – ERSPAN session ID
- Timestamp – timestamp of the mirrored packet
- SGT – Security Group Tag of the monitored frame (set to 0x0)
- P – This bit indicates that the ERSPAN payload is an Ethernet protocol frame (set to 0x1)
- FT – Frame Type (set to 0x0)
- Hw ID – Unique identifier of an ERSPAN engine within a system
- D (Direction) – Indicates whether the original frame was SPAN’ed in ingress or in egress. Ingress (0) or egress (1).

GRA (Timestamp Granularity) – Time unit to be supported for timestamping (Set to 0x2):
  - 00b – granularity = 100 microseconds
  - 01b – granularity = 100 nanoseconds
  - 10b – granularity = IEEE 1588
- O (Optional Sub-header) – The O flag indicates whether or not the optional platform-specific sub-header is present (set to 0x1).
- Platform ID – Platform identifier that needs to be recognized in order to parse the optional platform specific sub-header which follows (set to 0x5).
- Switch ID – platform specific information, switch ID
- Port ID – platform specific information, port ID of the monitored port
- Timestamp – timestamp upper 4 octets

38.7.1 SOC Properties

None

38.7.2 Configuration Flow

1. To create an IPv4 or IPv6 GRE tunnel, follow Step 1 in Section 38.6, ERSPANv2 Tunnel.
2. Create an ERSPANv3 tunnel by calling `bcm_tunnel_initiator_create(unit, intf, tunnel);`

- `tunnel`
  - `type = bcmTunnelTypeErspan`
  - `l3_intf_id` – Interface ID of IPv4 GRE tunnel created above (returned in intf-> l3a_intf_id in step 1)
  - `flags = BCM_TUNNEL_INIT_ERSPAN_TYPE3`
  - `hw_id` – HW ID field in the ERSPANv3 header
  - `switch_id` – Platform specific information, switch ID field in the ERSPANv3 header

3. To create a mirror profile, Follow Step 3 in Section 38.6, ERSPANv2 Tunnel.
4. To set header editing information for the created mirror profile, follow Step 4 in Section 38.6, ERSPANv2 Tunnel.

38.7.3 Application Reference

Example of ERSPANv3 configuration sequence.

- **Type:** CINT reference
- **Path:** `$SDK/src/examples/dnx/cint_dnx_mirror_example.c:mirror_erspan_example_v3`

38.8 Lawful Interception

The LI encapsulation contains the fields in the following table.

Table 94: LI Encapsulation Fields

Header Layer	Sub-Field	Size (Bits)	Comment
LI Tunnel	Content ID	32/64	Configured value from user
UDP	Checksum	16	Checksum = 0
	Length	16	Length = SNIFF copy length + 8 (UDP header) + 4 (content ID)
	Dst port	16	Configured value from user
	Src port	16	Configured value from user
IPv4	DIP	32	Configured value from user
	SIP	32	Configured value from user
	checksum	16	Compute per packet
	protocol	8	Protocol=17
	TTL	8	Configured value from user
	frame offset	13	Frame offset = 0
	flags	3	Flags = 0
	identification	16	Identification = 0
	total length	16	Total length = SNIFF copy length + 8 (UDP header) + 4 (content ID)
	TOS	8	TOS[7:2:] DSCP, configured from user TOS[1:0]: ECN = 0
	IHL	4	IHL = 5
	Version	4	Version = 4
IPv6	DIP	128	Configured value from user
	SIP	128	Configured value from user
	Hop Limit	8	Configured value from user
	Next Header	8	Next Header = 17
	Payload Length	16	Payload Length = SNIFF copy length + 8(UDP header) + 4(content ID)
	Flow Label	20	Auto-generate
	Traffic Class	8	Configured value from
	Version	4	6

The Ethernet header can be removed from the SNIFF copy according to setting.

NOTE:

- The `flow_label` field in the IPv6 header in the new encapsulation cannot be configured.

38.8.1 SOC Properties

None

38.8.2 Configuration Flow

1. Create an IPv4 or IPv6 tunnel with UDP by calling `bcm_tunnel_initiator_create(unit, bcm_l3_intf_t * intf, bcm_tunnel_initiator_t *tunnel)`.
   - tunnel
     - `flags=0` – Support other flag for IP tunnel
     - `type` – `bcmTunnelTypeUdp` or `bcmTunnelTypeUdp6`
     - `dip` – Configured DIP for IPv4
     - `sip` – Configured SIP for IPv4
     - `dip6` – Configured DIP for IPv6
     - `sip6` – Configured SIP for IPv6
     - `dscp` – Configured DSCP
     - `dscp_sel` – `bcmTunnelDscpAssign`
     - `ttl` – Configured TTL
     - `l3_intf_id` – ID of next interface
     - `udp_dst_port` – Configured UDP destination port
     - `udp_src_port` – Configured UDP source port
     - `flow_label` – Must set to 0 when `type=bcmTunnelTypeUdp6`
2. Create an LI tunnel by calling: `bcm_tunnel_initiator_create(unit, bcm_l3_intf_t * intf, bcm_tunnel_initiator_t *tunnel);`
   - `tunnel.flags` – Following flags supported: `BCM_TUNNEL_WITH_ID` and `BCM_TUNNEL_REPLACE/ BCM_TUNNEL_INIT_WIDE`. If `BCM_TUNNEL_INIT_WIDE` is set, the content ID of the lawful interception tunnel is 64 bits. Content ID = {`tunnel.aux_data2, tunnel.aux_data`}.
   - `tunnel.type` = `bcmTunnelTypeLawfulInterception`
   - `tunnel.aux_data` – Store content ID
   - `tunnel.tunnel_id` – Global ID of create LI tunnel
   - `tunnel.l3_intf_id` – ID of next interface (UDP_IPvx tunnel interface)
3. Create a mirror destination and set it by calling `bcm_mirror_destination_create(unit, bcm_mirror_destination_t * mirror_dest) and bcm_mirror_header_info_set(unit, flag, bcm_gport_t mirror_dest_id, bcm_mirror_header_info_t *mirror_header_info)`:
   - `mirror_dest.gport` - mirror dest port
   - `mirror_header_info.pp.eth_header_remove` – 1
   - `mirror_header_info.pp.bytes_to_remove`
   - `mirror_header_info.tm.out_vport` – Destination outlif(LI tunnel ID)
   - `mirror_header_info.tm.src_sysport` – Source port

   For more information, see Section 38.3, Mirror Header Generation.
4. Trigger Mirror-Action. See the PMF create mirror profile in Section 38.9, Trigger Mirror Action.
   - To remove the Ethernet header from SNIFF copy under the JR1 system header mode, the PMF preselector or qualifier should include `bcmFieldQualifyVlanFormat` to select `mir_dest`. More `mir_dest` should be created for tag format: untag/single/double tags.

38.8.3 Application Reference

An example of the Lawful Interception configuration sequence is available in the following reference:

- **Type:** CINT reference
- **Path:** `$SDK/src/examples/dnx/lawful_interception/cint_dnx_lawful_interception.c`

38.9 Trigger Mirror Action

Once a mirror profile is created (and points to a mirror tunnel if necessary), a packet can be associated with a mirror profile by using a <port, vlan> pair, field processor, or the trap code mechanism. Packets that need to be mirrored can be assigned with a mirror profile by one of the mechanisms.

38.9.1 SOC Properties

None

38.9.2 Configuration Flow

1. Assign a mirror profile to a packet by using a <port, vlan> pair calling
   `bcm_mirror_port_vlan_destination_add(unit, port, vlan, flags, destid, options);`
   Detailed parameter definitions of this API are described in the Sniff chapter of the *Traffic Manager Programming Guide*.
2. To assign a mirror profile to a packet by field processor, do the following:
   – For inbound mirroring, assign the packet with an `fp` action of `bcmFieldActionMirrorIngress`. Use mirror gport as the parameter (`mirror_dest` – `mirror_dest_id` returned by `bcm_mirror_destination_create`). For ERSPAN, use session ID as the mirror action qualifier
   – For outbound mirroring, assign the packet with an `fp` action of `bcmFieldActionMirrorEgress`. Use mirror profile as the parameter (`BCM_GPORT_MIRROR_GET` to get the mirror profile from a mirror gport).
3. Part of the trap can be assigned with a mirror profile directly. To assign a mirror profile to those traps, set the `mirror_profile` in `bcm_rx_trap_config_t` when configuring the traps. For more information on how to configure the traps, refer to Section 12.3, Trap Configuration.

38.9.3 Application Reference

Example of configuring mirror action through field processor.

- **Type:** CINT reference
- **Path:**
  - `$SDK/src/examples/dnx/field/cint_field_mirror.c`
  - `$SDK/src/examples/dnx/field/cint_field_egress_mirror.c`

38.9.4 API Descriptions

API Name	Highlights
`bcm_mirror_header_info_set()`	Set mirror system header
`bcm_mirror_header_info_get()`	Get mirror system header

Chapter 39: RFC 2544 Reflector

39.1 Introduction

Reflector flows are a set of tests defined in RFC 2544, *Benchmarking Methodology for Network Interconnect Devices*.

These tests can be used to describe the performance characteristics (such as service setup, fault management, and monitoring) of a network interconnecting device.

Generally, reflector flows identify packets requiring reflection, swaps the destination and source addresses on identified packets, and transmits them back to the sender.

There are two types of reflectors:

- **Internal** – Emulates a mode where the output interface of the system is looped-back, sending all packets that are supposed to egress the system back to the same interface through another pass in the system. Packets are expected to pass through the packet processing pipeline twice and are subjected to all applicable processing and filtering. Forwarding after the loopback is performed through the standard forwarding plane.
- **External** – Packets are looped back immediately on the input interface without going through the packet processing pipeline.

The reflector applications described in this section can be classified according to a number of parameters:

- Network layer (L2, L3, L4)
- Internal reflector or external reflector
- Packet cast (unicast or multicast)

The supported reflector applications are:

- L2 external unicast
- L2 external multicast
- L2 internal unicast
- L2 internal multicast
- L3 internal unicast (which optionally swap UDP source and destination)

The following reflector applications are not supported:

- L3 and L4 external unicast
- L3 and L4 multicast (both internal and external)

39.2 Application Configuration Checklist

39.2.1 L2 External Unicast Reflector

- Unicast reflector OutLIF – Allocate OutLIF of type unicast reflector
- L2 external unicast reflector trap – Allocate ingress trap and configure it to override the OutLIF to be the unicast reflector OutLIF.
- Field processor (ACL) group – Create a field processor group that identifies the flow by a user specific ACL. The group should do the following:
  - Override the forward action (trap code) making it a reflector trap previously allocated
  - Override the destination port to be the source system port
  - Add entries mapping user specific ACL to the allocated reflector trap

39.2.2 L2 External Multicast Reflector

- L2 external multicast reflector OutLIF – Allocate OutLIF of type external multicast reflector and specify the source MAC of the reflected packet
- L2 external multicast reflector trap – Allocate ingress trap
- Field processor (ACL) group – Create a field processor group that identifies the flow by a user specific ACL. The group should do the following:
  - Override the forward action (trap code) to be reflector trap previously allocated
  - Override the the destination port to be the source system port
  - Override the OutLIF to be a external multicast reflector OutLIF
  - Add entries mapping user specific ACL to the allocated reflector trap and OutLIF.

39.2.3 L2 Internal Unicast Reflector

The L2 internal unicast reflector supports recycling the packet either from the `recycle_mirror` port or from an external interface (such as a PHY loopback) as:

- One-pass: The packet is recycled through an external interface and appears as if the packet goes out of the device.
- Two-pass: The packet is recycled through the `recycle_mirror` port.

The L2 internal unicast reflector behavior is as follows:

- Recycle port – Allocate a recycle port for all packets that use a PTCH2 header.
- L2 internal reflector OutLIF – Allocate an OutLIF of type L2 internal unicast reflector.
- Field processor (ACL) group – Create a field processor group that identifies the flow by a user-specific ACL. The group should do the following:
  - Override the OutLIF to be the allocated reflector OutLIF.
  - Optionally set the ACE context value to support IP or UDP swap.
  - Override the destination port to be the recycle port for loopback.

  Add entries by mapping a user-specific ACL to the allocated reflector trap.

39.2.4 L2 Internal Multicast Reflector

- Recycle-Mirror port (skip it for one-pass operations)
  - Allocate recycle-mirror ports (recycle-mirror port for each local port).
  - Enable lossless priority context for each mirror-recycle port
  - Recycle mirror ports support two priorities contexts (lossless and high). L2 internal multicast reflector application requires lossless context to be allocated

- L2 internal reflector OutLIF – Allocate OutLIF of type L2 internal multicast reflector and specify the source MAC of the reflected packet.
- Field processor (ACL) group: create a field processor group that identifies the flow by a user specific ACL. The group should do the following:
  – Override the OutLIF to be the allocated reflector OutLIF

39.2.5 L3 Internal Unicast Reflector

- Recycle port – Allocate a recycle port for all packets that use a RCH header.
- Unicast reflector OutLIF – Allocate an OutLIF of type unicast reflector.
- Recycle OutLIF – Allocate OutLIF of type RCH.
- Field processor (ACL) group – Create a field processor program that identifies the flow by a user-specific ACL. The program should do the following:
  – Override OutLIF0 to be unicast reflector OutLIF.
  – Override OutLIF1 to be RCH OutLIF.
  – Override PP-DSP to be the TM port of the allocated recycle port.
  – Optionally set context-value to enable UDP swap.
  – Optionally reset TTL.

  Add entries mapping a user-specific ACL to the allocated unicast reflector OutLIF, RCH OutLIF, and TM recycle port.

39.3 L2 External Unicast Reflector

The packet is identified by a user-specific ACL in the ingress device. In such cases, SA and DA in Ethernet layer are swapped and the packet is looped back to the source port.

External reflectors emulates a mode in which the packet looped backed immediately and therefore, additional packet processing such as VLAN editing does not occur.

39.3.1 SOC Properties

None

39.3.2 Configuration Flow

1. Unicast reflector OutLIF – To allocate OutLIF for unicast reflector call the API

   `bcm_switch_reflector_create(unit, flags &encap_id, &data)`

   – `flags` – To allocate a specific OutLIF, set to `BCM_SWITCH_REFELCTOR_WITH_ID` (otherwise 0). Use `BCM_SWITCH_REFELCTOR_REPLACE` to modify data.
   – `encap_id` – (IN) Specify the required OutLIF when the flag `BCM_SWITCH_REFELCTOR_WITH_ID` is set. (OUT) the allocated OutLIF.
   – `data.type` – Set to `bcmSwitchReflectorUc`
   – `data.encap_access` – Use the default `bcmEncapAccessNativeArp(0)` or `bcmEncapAccessTunnel1` value:
     - `bcmEncapAccessNativeArp`: phase 2
     - `bcmEncapAccessTunnel1`: phase 3
2. L2 external unicast reflector trap:
   a. Allocate custom trap code by calling the API `bcm_rx_trap_type_create(unit, flags, type, &trap_id)`
      - `flags` – Set to 0

- `type` – Set to `bcmRxTrapUserDefine`
- `trap_id` – Is set to the allocated trap ID

b. Set trap to override the OutLIF to be the unicast reflector OutLIF by calling the API

`bcm_rx_trap_set(unit, trap_id, &config)`

- `trap_id` – Set to trap ID previously allocated
- `config.encap_id` – Set to the reflector OutLIF gport. Converting from OutLIF interface (which is the output of the API `bcm_switch_reflector_create)` to the OutLIF gport done by the macro `BCM_L3_ITF_LIF_TO_GPORT_TUNNEL`
- `config.flags` – Set to `BCM_RX_TRAP_UPDATE_ENCAP_ID`

For general details about traps, refer to Chapter 12, Traps.

3. Field Processor group for L2 external unicast reflector. For general details about field processor, refer to Chapter 11, Field Processor. For more information, refer to the CINT example.

39.3.3 Shell Commands

None

39.3.4 Application Reference

CINT example with detailed description of the application

- **Type:** CINT reference
- **Path:** `$SDK/src/examples/dnx/reflector/cint_reflector_l2_external.c`

NOTE: The main functions of the application `cint_reflector_l2_external_uc_create()` are:

- To create the required program, follow the reference in file `cint_reflector_l2_external.c`, function `cint_reflector_l2_external_fp_create()`
- To add entries follow the reference in `cint_reflector_l2_external.c` function `cint_reflector_l2_external_fp_entry_add()`

39.4 L2 External Multicast Reflector

The packet is identified be user specific ACL at ingress device. The DA is replaced by the original SA. The SA is replaced with a configurable MYMAC (per allocated OutLIF) and the packet is looped back to source.

External reflectors emulate a mode in which the packet is looped backed immediately and therefore, additional packet processing such as VLAN editing does not occur.

39.4.1 SOC Properties

None

39.4.2 Configuration Flow

1. Multicast external reflector OutLIF – To allocate OutLIF for multicast external reflector call the API `bcm_switch_reflector_create(unit, flags &encap_id, &data)`
   – `flags` – To allocate a specific OutLIF, set to `BCM_SWITCH_REFELCTOR_WITH_ID` (otherwise 0). Use `BCM_SWITCH_REFELCTOR_REPLACE` to modify data.

- `encap_id` – (IN) Specify the required OutLIF when the flag `BCM_SWITCH_REFELCTOR_WITH_ID` is set. (OUT) the allocated OutLIF.
- `data.type` – Set to `bcmSwitchReflectorL2McExternal`
- `data.mc_reflector_my_mac` – Specify the source MAC of the reflected packet
- `data.encap_access` – Use the default `bcmEncapAccessNativeArp(0)` or `bcmEncapAccessTunnel1` value:
  - `bcmEncapAccessNativeArp`: phase 2
  - `bcmEncapAccessTunnel1`: phase 3

2. L2 external multicast reflector trap.
   a. Allocate custom trap code by calling the API `bcm_rx_trap_type_create(unit, flags, type, &trap_id)`
      - `flags` – Set to 0
      - `type` – Set to `bcmRxTrapUserDefine`
      - `trap_id` – Is set to the allocated trap ID
   b. Set trap by calling the API `bcm_rx_trap_set(unit, trap_id, &config)`
      - `trap_id` – Set to trap ID previously allocated
      - `config.flags` – Set to 0

      For general details about traps, refer to Chapter 12, Traps.
3. Field Processor group for L2 external multicast reflector. For general details about field processor, refer to Chapter 11, Field Processor. For more information, refer to the CINT example.

39.4.3 Shell Commands

None

39.4.4 Application Reference

CINT example with detailed description of the application.

- **Type:** CINT reference
- **Path**: `$SDK/src/examples/dnx/reflector/cint_reflector_l2_external.c`

NOTE: The main functions of the application `cint_reflector_l2_external_mc_create()` are:

- To create the required program follow the reference in `cint_reflector_l2_external.c`, function `cint_reflector_l2_external_fp_create()`
- To add entries follow the reference in `cint_reflector_l2_external.c`, function `cint_reflector_l2_external_fp_entry_add()`

39.5 L2 Internal Unicast Reflector

The packet is identified by a user-specified ACL at the egress device. The SA and DA in the Ethernet layer are always swapped. The IP address and UDP port can optionally be swapped if the IP and UDP header exist according to the user's configuration in the ACL. The terminated header will be terminated before recycling to the second pass. An IVE or EVE action on the packet in the first pass can also be supported.

NOTE: The IPv6 header contains an extension header it cannot support. It cannot support the reflector on LAG that contains the CoE port for an IPv6 format packet.

39.5.1 SOC Properties

None.

39.5.2 Configuration Flow

In the L2 internal unicast reflector, packets can go into the pipeline for the second pass based on the Recycle Mirror port (two-pass L2 internal reflector) or an external interface (such as a PHY/MAC loopback without the ptch2 header, also known as a one-pass L2 internal reflector).

The IP address (SIP/DIP) and UDP port (DPORT/SPORT) in the packet can optionally support swap in the first pass. The MAC address (DA/SA) always swaps.

1. Recycle-Mirror ports – For more details about creating a recycle-mirror port, refer to the Sniff chapter in the *Traffic Manager Programing Guide* (88690-PG2xx).
   - If the L2 internal unicast reflector application is based on a non `recycle_port` (one-pass), such as through a PHY loopback, skip this step.
   - Calling the `bcm_mirror_port_to_rcy_port_map_get()` API to get a recycled mirror port according to the forward port in the first pass. Then the packet goes into the pipeline for the second pass through the recycle-mirror port.
   - Calling the `bcm_mirror_port_to_rcy_port_map_set()` API to unmap the forward port from the recycle-mirror port first by setting `rcy_map_info.rcy_mirror_port=BCM_PORT_INVALID`.
   - Calling the `bcm_mirror_port_to_rcy_port_map_set()` API to map the forward port to the recycle-mirror port in a high priority recycle context by setting `rcy_map_info.priority_bitmap = BCM_MIRROR_PORT_TO_RCY_MAP_INFO_PRIORITY_HIGH`.
   - Normal priority can be set on Recycle Mirror to support the TM shape:
     - Calling the `bcm_mirror_port_to_rcy_port_map_set()` API with `BCM_MIRROR_PORT_TO_RCY_MAP_INFO_PRIORITY_HIGH`
     - Calling the `bcm_switch_control_set (uint, bcmSwitchReflectorL2IntPriority, value)` API to set recycled packet's priority in the first pass, where value 0 is high and value 1 is normal. The default value is 0. Setting the recycled packets' priority is global. This API changes all recycled packet's priority if carried with the PTCH2 header.
2. Unicast internal reflector OutLIF – To allocate OutLIF for unicast internal reflector, call the API:

   `bcm_switch_reflector_create(unit, flags &encap_id, &data)`
   - `flags` – Set to `BCM_SWITCH_REFELCTOR_WITH_ID` to allocate a specific OutLIF, or set to 0 otherwise. Use `BCM_SWITCH_REFELCTOR_REPLACE` to modify data.
   - `encap_id` – (IN) Specify the required OutLIF if the flag `BCM_SWITCH_REFELCTOR_WITH_ID` is set. (OUT) the allocated OutLIF.
   - `data.type` – Set to `bcmSwitchReflectorL2UcInternal`
   - `data.encap_access` – Use the default `bcmEncapAccessNativeArp(0)` or `bcmEncapAccessTunnel1` value:
     - `bcmEncapAccessNativeArp`: phase 2
     - `bcmEncapAccessTunnel1`: phase 3
3. Field Processor group for L2 internal unicast reflector: Chapter 11, Field Processor, for general details about the field processor. See also the CINT example for more information. The option to swap the IP or UDP header is controlled by setting the ACE context value:
   - `bcmFieldAceContextReflectorL2IntIpUdpSwap` – Swap the IP address and UDP port, if they exist.
   - `bcmFieldAceContextReflectorL2IntIpSwap` – Swap the IP address, if it exists.
   - `bcmFieldAceContextReflectorL2Int` – Do not swap the IP address and UDP port.

4. Reflector OutLIF set by the field processor should be action:
   – `bcmFieldActionOutInterface1` – The `bcmFieldActionOutInterface0` action holds OutAC. Do not put the reflector OutLIF to `bcmFieldActionOutInterface0` if EVE is applied.

39.5.3 Shell Commands

None.

39.5.4 Application Reference

CINT Example with detailed description of the application

- **Type:** CINT reference
- **Path:** `$SDK/src/examples/dnx/reflector/cint_reflector_l2_internal.c`

The main function of the application is `cint_reflector_l2_internal_uc_create()`. To create the required program, follow the reference in the `cint_reflector_l2_internal.c` function:

- `cint_reflector_l2_internal_uc_create()` – Recycle with a RCY_MIR port.
- `cint_reflector_l2_internal_uc_with_dedicated_rcy_create` – Recycle with a dedicated RCY port.
- `cint_reflector_l2_internal_uc_non_rcy_port_create` – Without a recycle port, the packet recycles with a MAC/PHY loopback.

To add entries, follow the reference in the `cint_reflector_l2_internal_uc_fp_entry_add()` function in the `cint_reflector_l2_internal.c` file.

39.6 L2 Internal Multicast Reflector

The packet is identified be a user-specific ACL in the egress device. The DA is replaced by the original SA. The SA is replaced by a configurable MYMAC and the packet looped back to source. An IVE or EVE action on the packet in the first pass can also be supported.

NOTE: The L2 internal multicast reflector cannot support swapping for an IPvX address and UDP port. It cannot support the reflector on LAG that contains a CoE port for a IPv6 format packet.

39.6.1 SOC Properties

None

39.6.2 Configuration Flow

1. Recycle-Mirror ports – For general details about creation of recycle-mirror port, refer to the section *Sniff* in the BCM88690 *Traffic Manager Programing Guide*.
   – If the L2 internal multicast reflector application is based on a non `recycle_port` (one-pass), such as through the PHY loopback, skip this step.
   – Call the API `bcm_mirror_port_to_rcy_port_map_get()` to get a recycled mirror port according to the forward port in the first pass. Then, the packet goes into the pipeline for the second pass through the recycle mirror port.
   – Call the API `bcm_mirror_port_to_rcy_port_map_set()` to unmap the forward port from the recycle mirror port first by setting `rcy_map_info.rcy_mirror_port=BCM_PORT_INVALID`.

  - Call the API `bcm_mirror_port_to_rcy_port_map_set()` to map the forward port to the recycle mirror port in a high-priority recycle context by setting `rcy_map_info.priority_bitmap = BCM_MIRROR_PORT_TO_RCY_MAP_INFO_PRIORITY_HIGH`

2. Multicast external reflector OutLIF – To allocate OutLIF for multicast external reflector call the API `bcm_switch_reflector_create(unit, flags &encap_id, &data)`
   - `flags` – To allocate a specific OutLIF, set to `BCM_SWITCH_REFELCTOR_WITH_ID` (otherwise 0). Use `BCM_SWITCH_REFELCTOR_REPLACE` to modify data.
   - `encap_id` – (IN) Specify the required OutLIF when the flag `BCM_SWITCH_REFELCTOR_WITH_ID` is set. (OUT) the allocated OutLIF
   - `data.type` – Set to `bcmSwitchReflectorL2McInternal`. Set to `bcmSwitchReflectorL2McInternalOnePass` if the L2 internal MC reflector application is based on a non `recycle_port` (one-pass).
   - `data.mc_reflector_my_mac` – Specify the source MAC of the reflected packet.
   - `data.encap_access` – Use the default `bcmEncapAccessNativeArp(0)` or `bcmEncapAccessTunnel1` value:
     - `bcmEncapAccessNativeArp`: phase 2
     - `bcmEncapAccessTunnel1`: phase 3
3. Field Processor group for L2 internal multicast reflector. For general details about field processor, refer to Chapter 11, Field Processor. For more information, refer to the CINT example. The ACE context value for multicast is `bcmFieldAceContextReflectorL2IntMc`.
4. Reflector OutLif set by the field processor should be action:
   - `bcmFieldActionOutInterface1` – The `bcmFieldActionOutInterface0` action holds OutAC. Do not put the reflector OutLif to `bcmFieldActionOutInterface0` if EVE is applied.

39.6.3 Shell Commands

None

39.6.4 Application Reference

CINT example with detailed description of the application.

- **Type:** CINT reference
- **Path:** `$SDK/src/examples/dnx/reflector/cint_reflector_l2_internal.c`

NOTE: The main functions of the application `cint_reflector_l2_internal_mc_create()` are:

- To create the required program follow the reference in `cint_reflector_l2_internal.c`, function `cint_reflector_l2_internal_fp_create()`
- To add entries follow the reference in `cint_reflector_l2_internal.c`, function `cint_reflector_l2_internal_fp_entry_add()`

39.7 L3 Internal Unicast Reflector

A packet is identified be a user-specific ACL at the egress device, SIP and DIP are swapped, L4 swap (UDP ports) and TTL reset are optional, and the packet is looped back to source.

39.7.1 SOC Properties

Require an additional recycle port, for example:

```
tm_port_header_type_in_211=RCH_0
ucode_port_211=RCY0.0:core_0.211
```

39.7.2 Configuration Flow

1. Recycle port – For details, see Section 17.2, Recycle Port.
2. Unicast reflector OutLIF – To allocate OutLIF for unicast reflector call the API `bcm_switch_reflector_create(unit, flags &encap_id, &data)`
   - `flags` – To allocate a specific OutLIF, set to `BCM_SWITCH_REFELCTOR_WITH_ID` (otherwise 0). Use `BCM_SWITCH_REFELCTOR_REPLACE` to modify data.
   - `encap_id` – Specify the required OutLIF when the flag `BCM_SWITCH_REFELCTOR_WITH_ID` is set. (OUT) the allocated OutLIF.
   - `data.type` – Set to `bcmSwitchReflectorUc`.
   - `data.encap_access` – Use the default `bcmEncapAccessNativeArp(0)` or `bcmEncapAccessTunnel1` value:
     - `bcmEncapAccessNativeArp`: phase 2
     - `bcmEncapAccessTunnel1`: phase 3
3. Recycle OutLIF – For details, see Section 17.4, Encapsulation Recycle Entry. See the CINT example `create_recycle_entry_with_defaults()`. Call the API `bcm_l2_egress_create(unit, &recycle_entry)` to create a recycle OutLIF entry with following parameter:
   - `recycle_entry.flags` – `BCM_L2_EGRESS_RECYCLE_HEADER`
   - `recycle_entry.recycle_app` – `bcmL2EgressRecycleAppDropAndContinue`
4. Field Processor group for L3 internal unicast reflector. The optional TTL reset can be achieved by adding to the Field Processor program an action to override the TTL field (action type `bcmFieldActionTtlSet`). The CINT example includes an input parameter which allows either performing a UDP swap (or not) through setting an ACE context value:
   - `bcmFieldAceContextReflector` – UDP swap
   - `bcmFieldAceContextNull` – No UDP swap

   For general details about field processor, see Chapter 11, Field Processor. For more information, refer to the CINT example.

39.7.3 Shell Commands

None

39.7.4 Application Reference

CINT example with detailed description of the application.

- **Type:** CINT reference
- **Path:** `$SDK/src/examples/dnx/reflector/cint_reflector_l3_internal.c`

NOTE: The main functions of the application `cint_reflector_l3_internal_uc_create()` are:

- To create the required program follow the reference in `cint_reflector_l3_internal.c`, function `cint_reflector_l3_internal_fp_create()`
- To add entries follow the reference in `cint_reflector_l3_internal.c`, function `cint_reflector_l3_internal_fp_entry_add()`

39.8 API Descriptions

39.8.1 Reflector OutLIF

API Name	Highlights
`bcm_switch_reflector_create()`	Allocate and set reflector OutLIF
`bcm_switch_reflector_destroy()`	Destroy allocated reflector OutLIF
`bcm_switch_reflector_traverse()`	Traverse all allocated reflector OutLIFs
`bcm_switch_reflector_get()`	Get reflector OutLIF data

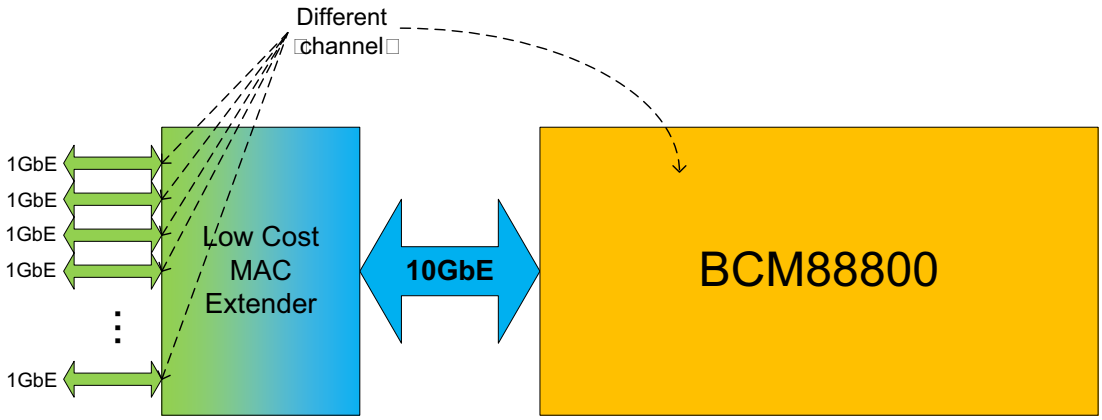
Chapter 40: Port Extender Channelization over Ethernet

40.1 Introduction

Channelization over Ethernet (CoE) is the channelization of a port to receive and transmit packets through the same physical interface. It is possible to aggregate multiple low-speed interfaces to a higher one. Compared to other channelized interfaces, such as Interlaken, a CoE channel is carried out through fields contained in the packet header without any interface channel information. The advantages of CoE make it widely accepted as a port extender in service provider devices. Port VLAN CoE is one type of CoE where channel information is carried in the VID of a CoE tag. In this chapter, CoE refers to Port VLAN CoE.

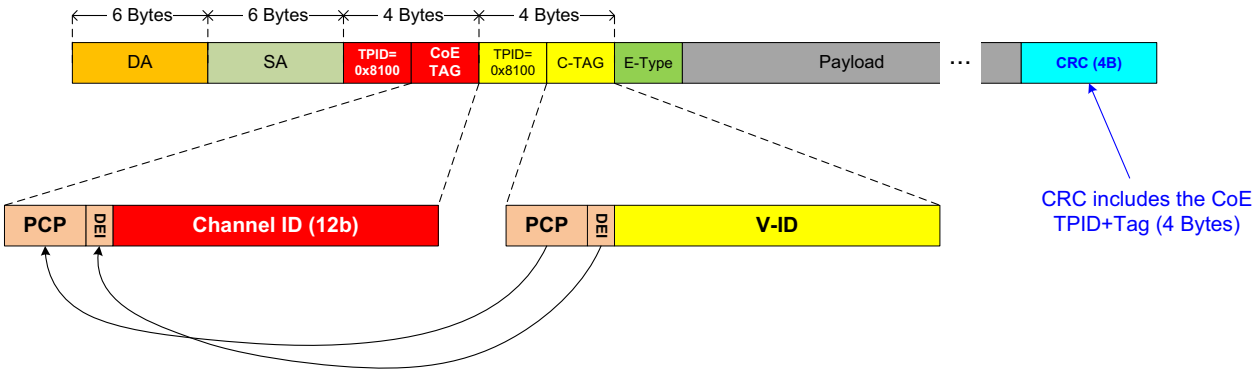
The following figure shows a system with a BCM88800 (J2C) device and an external MAC extender.

Figure 29: BCM88800 with External MAC Extender



The channelization is achieved by using an extra IEEE 802.1Q VLAN tag to specify the channel. The following figure shows the general packet format for all traffic of channelized interfaces. As the figure shows, the CoE tag is inserted at a fixed 12-byte offset to the first byte following SFD. The Channelization Tag is 4 bytes in length and is based on the format of an 802.1Q VLAN tag. The VID is used as the channel ID. The channel ID numbering starts at 0. For example, channel 0 is the first channel, and each physical interface has an individual numbering space for all channels belong to it.

Figure 30: Channelized Interface Packet Format



NOTE: COE flow control is not supported on the BCM88690 only. Only on the BCM88690, L2VPN applications (such as VXLAN, VPLS, EVPN, and so on) are not supported on a COE port.
The following applications are not supported with a COE port:

- TDM service cannot be sent directly to the COE port
- MACsec cannot be enabled on the COE port.

40.2 Application Configuration Checklist

The CoE application contains the following configuration sequences:

- CoE port settings
- Trap, TM, or Mirror a packet to CoE port
- CoE into LAG
- PCP_DEI source select
- Outbound mirror: remove CoE tag from the mirror copy
- CoE flow control
- Estimated bytes to add on a CoE port

40.3 CoE Port Settings

Multiple logical ports can be defined per physical interface. Each channel, defined by physical interface × VLAN-ID, receives a new CoE port that is presented by the local port. As with any other local port, the same settings apply to it, such as PP functionality of the Ethernet port, Source-System port configuration, and so on.

The following settings exist for a CoE port:

- Enable Logical-Port to be a CoE port.
- Map multiple physical-port × VLAN-ID to CoE port.

40.3.1 SOC Properties

Define a CoE port as `ucode_port_<xx>.BCM88690=<INTERFACE>.<yy>:core_<n>.<zz>`, where the parameters are as follows:

- `xx`: Local port of CoE port.
- `INTERFACE`: Interface of CoE port.
- `yy`: Channel ID of CoE port.
- `n`: Core ID of CoE port.
- `zz`: TM port of CoE port.

The following example creates two CoE ports for the same physical interface:

- `ucode_port_20.BCM8869X=XE14.0:core_0.100`
- `ucode_port_21.BCM8869X=XE14.1:core_0.101`

40.3.2 Configuration Flow

To enable a CoE port, call `bcm_port_control_set(unit, port, type, value);`

- `port`: CoE logical port
- `type`: `bcmPortControlExtenderEnable` for CoE feature
- `value`: 1 to enable, and 0 to disable

To map the CoE port, use the API `bcm_port_extender_mapping_info_set(unit, flags, bcmPortExtenderMappingTypePortVlan, mapping_info)`

- `flags`: Both `BCM_PORT_EXTENDER_MAPPING_INGRESS` and `BCM_PORT_EXTENDER_MAPPING_EGRESS` should be set.
- `type`: `bcmPortExtenderMappingTypePortVlan`
- `mapping_info`: Holds the mapping information.
  - Initialize the structure by calling `bcm_port_extender_mapping_info_t_init`.
  - `pp_port`: Not used. `pp_port` is obtained from the logical port.
  - `tunnel_id`: Not used.
  - `phy_port`: Logical port. Channel 0's TM port for an interface. The range is 0 to 255
  - `vlan`: VID of CoE port, both for ingress mapping and egress encapsulation.
  - `fc_channel_id`: Flow control mapped `channel_id`. For details, see Section 40.8, COE Flow Control Mapping..

The following example maps local port 20, physical interface XE14, channel 0, VLAN 20

```
print bcm_port_get(unit,20,&flags_get,&interface_info,&port_info);
mapping_info.phy_port = 20;
mapping_info.vlan = 20;
print bcm_port_extender_mapping_info_set(unit,flags,type, &mapping_info);
```

For getting the operation through `bcm_port_extender_mapping_info_get()`, the `phy_port` in `mapping_info` is different in the IN and OUT directions:

- IN: Logical port.
- OUT: `core_id` + `tm_port`,

The dynamic COE port configure sequence is as follows:

1. Enable the COE port
2. Call `bcm_port_control_set(uint,port,bcmPortControlExtenderEnable,1);`
3. Call `bcm_port_extender_mapping_info_set(unit, flags, bcmPortExtenderMappingTypePortVlan, mapping_info);`
4. Disable the COE port.
5. Call `bcm_port_control_set(uint,port,bcmPortControlExtenderEnable,0);`

NOTE: Disable the COE port before port removal on the TM layer. Disable the COE port on channel 0 last. Disabling COE ports that are not on channel 0 is allowed without changing the action of other COE ports. Remove the disabled port from this interface (PTC).
Dynamic COE settings cannot support traffic on-the-fly.

40.3.3 Shell Commands

None

40.3.4 Application Reference

For an example, refer to the following:

- **Type:** CINT reference
- **Path:** `$SDK/src/examples/dnx/extender/cint_dnx_coe_basic.c`

- **Description:** Configuration steps are in `port_dnx_coe_basic_service()`

40.4 Trap, TM, or Mirror to a CoE Port

When trapping, injection, and mirroring support is required for a CoE port (that is, with CoE tag encapsulation), additional configurations on ePMF are required:

- ePMF profile setting on CoE port to select on ePMF
- ePMF filters TM, trap, or mirror service to set the correct ACE context for them. If the COE port is set to the Raw header type, the ePMF rule is needed as Trap/Injection/Mirror/TDM. The ePMF context selection should be based on the forward type.

40.4.1 SOC Properties

None

40.4.2 Configuration Flow

Set an ePMF profile on CoE port by calling `bcm_port_class_set(unit, port, pclass, class_id);`

- `port`: CoE port.
- `pclass`: `bcmPortClassFieldEgressPacketProcessingPortCs`
- `class_id`: The PMF profile on the CoE port. It should match the ePMF setting.

The ePMF creates an FG to set the ACE context value with a pre-selector containing a *Forward Context* and *Port PMF* profile to filter the service.

The pre-selector qualifier type and value are as follows:

- `qual_type == bcmFieldQualifyPortClassPacketProcessing`
  - ePMF profile set in `bcm_port_class_set()`.
- `qual_type == bcmFieldQualifyContextId`
  - Injected TM: `bcmFieldForwardContextTm`.

    NOTE: On the BCM88690 only, `bcmFieldQualifyContextId` should be replaced by `bcmFieldQualifyPphPresent` under IOP mode.
  - Trap: `bcmFieldForwardContextIngressTrapLegacy`
  - Egress Mirror: `bcmFieldForwardContextMirrorOrSs`
  - Ingress Mirror: `bcmFieldForwardContextMirrorOrSs`
  - TDM: `bcmFieldForwardContextTdm`

FG action is `bcmFieldActionAceEntryId` with ACE adding as:

- `action_type` == `bcmFieldActionAceContextValue`
  - `bcmFieldAceContextTmToCOE`

For ingress mirroring only to the COE port, the mirror copy should be updated with the new system header through calling `bcm_mirror_header_info_set()` without setting the `BCM_MIRROR_DEST_EGRESS_ADD_ORIG_SYSTEM_HEADER` flag.

The dynamic CoE port configuration sequence is as follows:

1. Enable the CoE port.
2. Call `bcm_port_control_set(uint,port,bcmPortControlExtenderEnable,1);`

3. Call `bcm_port_class_get(unit,port, bcmPortClassFieldEgressPacketProcessingPortCs, &old_profile);`
4. Call `bcm_port_class_set(unit,port, bcmPortClassFieldEgressPacketProcessingPortCs, new_profile);`
5. Call `bcm_port_extender_mapping_info_set(unit, flags, bcmPortExtenderMappingTypePortVlan, mapping_info);`
6. Disable the CoE port.
7. Call `bcm_port_control_set(uint,port,bcmPortControlExtenderEnable,0);`
8. Call `bcm_port_class_set(unit,port, bcmPortClassFieldEgressPacketProcessingPortCs, old_profile);`

40.4.3 Shell Commands

None

40.4.4 Application Reference

- **Type:** CINT reference
- **Path:** `$SDK/src/examples/dnx/extender/cint_dnx_coe_basic.c`
- **Description:** For more configuration steps, refer to `cint_tm_to_coe_main_config()`
  - `app_type == 2`: Config for Mirror application
  - `app_type == 1`: Config for Trap application
  - `app_type == 0`: Config for TM application

40.5 COE Port into a LAG

A COE port can be added to or removed from a LAG (trunk group). The COE port should exist and be configured before adding it to or removing it from the LAG. When a COE port is removed from a LAG group, re-create all of its configurations.

40.5.1 SOC Properties

None

40.5.2 Configuration Flow

1. Enable the CoE port and configure mapping as described previously.
2. Create a LAG and add the COE port into the LAG as a member.
3. Remove the COE port from the LAG.

NOTE: On the BCM88690 only, when a COE port added/removed to/from a LAG group, reconfigure ePMF CS profile on this LAG group with API `bcm_port_class_set(unit, lag_gport, bcmPortClassFieldEgressPacketProcessingPortCs, pmf_profile)`

40.5.3 Shell Commands

None

40.5.4 Application Reference

- **Type:** CINT reference for COE port and trunk
- **Path:** `$SDK/src/examples/dnx/extender/cint_dnx_coe_basic.c`
- **Description:** For more configurations, refer to `cint_dnx_coe_trunk_service()` and `cint_dnx_coe_trunk_multicast_service()`.

40.6 PCP_DEI Source Select

PCP_DEI source supports selecting from a port attribute or `Nwk_QoS` for mapping PCP/DEI into a COE tag.

40.6.1 SOC Properties

None

40.6.2 Configuration Flow

1. Enable the CoE port and configure mapping as described previously in this chapter.
2. To set the PCP or DEI source, call `bcm_switch_control_port_set(unit, port, bcmSwitchTagPcpDeiSrc, arg)`:

- `arg`:
  - 3 – Port attribute
  - 6 – `Nwk_QoS.4msb`
  - 7 – `Nwk_QoS.4lsb`

3. Configure QoS mapping, which is either the default QoS mapping or customized QoS mapping.

40.6.3 Shell Commands

None

40.6.4 Application Reference

For default QoS mapping, refer to the following:

- **Type**: CINT reference for QoS mapping
- **Path**: `$SDK/src/appl/reference/dnx/appl_reg_qos_init.c`

40.7 Outbound Mirror: Remove COE Tag

Some applications request the removal of the COE tag for a mirror copy when a packet hits an outbound mirror rule on the COE port. The following are the two methods to accomplish this:

- Enable COE on the `Rcy_Mir` port. Other services using the same `Rcy_Mir` will break, such as internal reflector or OAM UPMEP.
- Set IVE in the mirror system header to remove the COE tag. The IVE action would not work depending on services.

40.7.1 SOC Properties

None.

40.7.2 Configuration Flow

Enable COE on the `Rcy_Mir` port.

1. Call the `bcm_mirror_port_to_rcy_port_map_get(unit, flags, coe_port_mir_src, rcy_map_info)` API to get the `rcy_mirror` port for the COE port.
2. Call the `bcm_port_control_set(unit, rcy_map_info.rcy_mirror_port, bcmPortControlExtenderEnable, value)` API to enable or disable COE tag removal on Rcy_Mirror port.
3. Call the `bcm_port_extender_mapping_info_set(unit, flags, type, *mapping_info)` API to add ingress mapping on the `Rcy_Mirror` port to support COE tag removal, if enabled.
   – First, initialize the structure by calling `bcm_port_extender_mapping_info_t_init`.
   – `mapping_info.vlan`: `coe_vlan`
   – `mapping_info.phy_port`: `rcy_mirror` port

Set IVE to mirror the system header.

1. Create IVE action to remove the outer tag (COE tag) for mirror copy.
2. Create an outbound mirror source from a COE port.
3. Calling API bcm_mirror_header_info_set(unit, flags, mirror_dest_id, mirror_header_info) to set IVE action
   – `flags`: `BCM_MIRROR_DEST_STAMP_IVE_ACTION_ID`
   – `mirror_header_info.pp.ingress_vlan_translate_action_id`: IVE action ID

40.7.3 Shell Commands

None

40.7.4 Application Reference

For additional configure steps, refer to `port_dnx_rm_coe_tag_egress_mirror()`.

- **Type**: CINT reference
- **Path**: `$SDK/src/examples/dnx/extender/cint_dnx_coe_basic.c`

40.8 COE Flow Control Mapping

COE flow control is not supported on the BCM88690 only.

COE flow control works through `fc_channel_id` to the Flow-Control HW block. The `fc_channel_id` value is mapped by PTC + VID as follows:

- PTC: `tm_port` of channel 0. Up to 6 bits. The range is from 0 to 63.
- VID: Up to 6 bits. The range is from 0 to 63.
- `fc_channel_id`: The range is from 0 to 255.

NOTE: For PAUSE mode, each COE port is mapped to a `fc_channel_id` value, which is one of 256 channels per core. For PFC mode, each COE port updates four channels at a time because they are bound together to support four priorities at once (the range is 0 to 63).

`fc_channel_id` must be configured when adding COE port mapping.

40.8.1 SOC Properties

The SOC properties are as follows:

- `port_priorities=8` – Set the number of priorities per port.
- `fc_calendar_coe_mode=PAUSE` or `PFC`
- `fc_coe_mac_address=01:80:C2:00:00:01`
- `fc_coe_data_offset=16/18`, 16 for PAUSE mode and 18 for PFC mode.
- `fc_coe_ethertype=0x8808`
- `fc_calendar_pause_resolution=8`

40.8.2 Configuration Flow

1. Call the `bcm_port_control_set(unit, coe_port, bcmPortControlExtenderEnable, value)` API to enable or disable the COE port.
2. Call the `bcm_port_extender_mapping_info_set(unit, flags, type, *mapping_info)` API to add mapping for a COE port.
   - First, initialize the structure by calling `bcm_port_extender_mapping_info_t_init`.
   - `mapping_info.vlan: coe_vlan`
   - `mapping_info.phy_port: logical port.`
   - `mapping_info.fc_channel_id`: Mapped `fc_channel_id`. The range is 0 to 255 for PAUSE mode and 0 to 63 for PFC mode. Set to –1 if without flow control.

40.8.3 Shell Commands

None.

40.8.4 Application Reference

For additional configure steps, refer to `port_dnx_coe_config_with_fc()`.

- **Type**: CINT reference
- **Path**: `$SDK/src/examples/dnx/extender/cint_dnx_coe_basic.c`

Chapter 41: Segment Routing over IPv6

41.1 Introduction

Segment routing over IPv6 (SRv6) is segment routing technology over the IPv6 dataplane, using a routing extension header (SRH).

An SRv6 ingress node bridges and routes based on packet information and steers packets through an ordered list of instructions, called segments or segment identifiers (SIDs). Each instruction represents a function called at a specific location in the network.

DNX devices support SRv6 classic SID (basic), micro SID (uSID), and generalized SID (GSID) formats. The DNX devices also support any combination of these formats in the SRv6 SID lists.

An *endpoint* node is the node that receives the SRv6 packet, and the IPv6.DIP is equal to its local interface IPv6 address. The endpoint node then forwards the packets toward the next SID in the SID list. The endpoint node can also terminate the SRv6 extension if it is configured as an endpoint penultimate segment pop (PSP) node.

The egress node is responsible for the SRv6 termination and bridges and routes based on the inner packet. DNX devices support both ultimate segment pop (USP) and ultimate segment decapsulation (USD) flavors.

This chapter focuses on the SRv6 objects and APIs that set the SRv6 applications this introduction describes.

The dedicated BCM SDK *srv6* module defines SRv6 objects and dedicated databases. The BCM SDK *tunnel* module is also used in several APIs.

41.1.1 SRv6 Segment ID Formats

SRV6 has several standard SID formats, and DNX devices can process the following SID formats:

- Classic (128b SID)
- Micro-SID (uSID)
- Generalized SID (GSID)

For the ingress node process (encapsulating an SRv6 header), the flow is agnostic to the SID format. The SRv6 SID encapsulation is a general flow that contains SIDs as raw data (128 bits per SID) that are set by the user.

For the egress node (terminating an SRv6 header), there is no difference between the formats other than the My-DIP classification. However, after the packet is identified as an egress node, the termination process is similar for all SID formats.

The only difference between the flows is for the endpoint node (see Section 41.2.1, Endpoint Node).

41.1.1.1 Classic SID

In Classic SID, there is no aggregation of multiple node information on the SID. In other words, each SID (128b) represents a node (location, function, or both) in the SRv6 topology.

For Classic SID, the SDK supports three types of structures:

- Locator (96 bits), function (16 bits), and argument or padding (16 bits)
- Locator (64 bits), function (16 bits), and argument or padding (48 bits)
- Full 128-bit classification – No locator and function separation.

41.1.1.2 Compressed SID Formats

The SIDs in an SRv6 domain are allocated from an address block called SID space. SIDs allocated from the same SID space share a common prefix.

DNX devices support two compressed SIDs formats: micro-SID (uSID) and Generalized Segment ID (GSID).

41.1.1.2.1 uSID

uSID is an SID format that contains a prefix and several micro-SIDs. Each micro-SID is a 16-bit SID that identifies a node in the SRv6 topology.

For uSID, the IPv6.DIP is constructed with a common prefix and 16-bit SIDs. The SDK supports a single structure type that consists of a 32-bit or a 48-bit prefix and up to six 16-bit uSIDs (depending on the prefix size). The configuration to work with uSID prefix of 32 bits or 48 bits is set with the following API:

```
bcm_switch_control_set(unit, bcmSwitchSrv6UsidPrefixBits, prefix_size <32/48>).
```

This is a global configuration that applies to all SRv6 USID tunnels.

NOTE: If using the DNX application reference code as-is, you can also make sure that the API is being called to configure the prefix size to 64 bits by setting the SoS property `appl_param_srv6_usid_prefix_48b_enable=1`.

Figure 31: uSID IPv6.DIP

The uSID does not contain a function or arg parameters, and it is not expected to have an SID with a uSID format as the VPN SID.

41.1.1.2.2 GSID

GSID is a compressed SID format. The GSID is a 32-bit value following the common prefix in the original SRv6 SID. A GSID container includes up to four GSIDs and is set as an SRv6 SID in the SID list.

For GSID, the IPv6 SID is constructed by a 48-bit or 64-bit common prefix followed by a single GSID segment, padding, and segment indicator (SI).

Figure 32: GSID IPv6.DIP

The common prefix and padding are not included in the SID list. Instead, the list consists of GSID containers where each container includes up to four GSIDs. This significantly reduces the size of the SID list in the SRH header.

GSID processing, therefore, engages the classical SRv6 Segment Left (SL) indication, as well as the segment index of the current GSID in the current SID.

As an SRv6 SID represents a GSID compressed container with up to four GSIDs. It is required to allow a dedicated compressible SID in the list to hold the prefix value, the initial GSID, and the initial SI value. This compressible SID is copied to the IPv6.DIP as a classical SRv6 SID, and it is used as the ingress node to the SRv6 GSID domain.

The configuration to work with a GSID prefix of 48 bits or 64 bits is set with the following API:

```
bcm_switch_control_set(unit, bcmSwitchSrv6GsidPrefixBits, prefix_size <48/64>)
```

This is a global configuration that applies to all SRv6 GSID tunnels.

NOTE: If using the DNX application reference code as-is, you can also make sure that the API is being called to configure the prefix size to 64 bits by setting the SOC property:

```
appl_param_srv6_gsid_prefix_64b_enable=1
```

For GSID compression, the 32b GSID is divided into to a locator and a function, where the locator is the 16 MSBs and the function is the 16 LSBs.

41.1.2 Definitions and Acronyms

Acronym	Definition
(e)BTA	(Estimated) bytes to add
(e)BTR	(Estimated) bytes to remove
EES	Egress encapsulation stack
EOC	End of compression
GSID	Generalized SID
InLIF	In Logical interface
LE	Last entry
OutLIF	Out Logical interface
RCH	Recycle header
RCH-EE	Recycle header of type extended encapsulation
RCY	Recycle
SI	Segment indication
SID	Segment identifier
SL	Segment left
SRH	SRv6 base header
uSID	Micro SID

41.2 Application Configuration Flow

To understand the SRv6 flow, read the following sections first:

- The basic packet processor concepts of switch and router abstraction information in the *Packet Processing Architecture Specification*, which provides an overview of the basic L2 and L3 objects.
- Section 22.6, ETH-RIF (L3 VSI Interface) and VRF Definition.
- Chapter 21, Ethernet Bridge
- Chapter 22, IP Router provides information about configurations that are related to next-hop information, namely FEC and Egress-ARP. Also, this chapter provides information relevant for the My-MAC operation.
- Chapter 30, IP Tunnel v6 Termination.
- Chapter 31, IP Tunnel v6 Encapsulation.
- Chapter 17, Standardized Recycle Header. This section provides information about the Recycle header (RCH) and its types.

41.2.1 Endpoint Node

This section discusses the following SRv6 functionalities:

- End.X, End.T – Fully covered.
- End.DX6, End.DX4, End.DT6, End.DT4, End.DT46, End.DX2, End.DX2V, End.DT2U, End.DT2M, End.B6.Encaps, End.B6.Encaps.Red – Partially covered here, and fully covered in Section 41.2.2, Ingress Node and Section 41.2.3, Egress Node.

SRv6 endpoint node is the node that performs the basic SRv6 functionality of swapping an SID in the SID list. When a packet of type AnyoSRv6oIPv6oETH arrives at an endpoint node, it is expected to have “My-DIP” match on the IPv6.DIP in the current node, and the packet editing will include the following steps:

1. Decrement SRv6.SL by 1.
2. Take the next SID from the SRv6.SID_list and place it in the IPv6.DIP.

NOTE:

- When the SRv6 packet is received with SRv6.SL=0, the endpoint node performs one of the Decapsulation functions (End.D*), which are described in Section 41.2.3, Egress Node.
- If the endpoint is a compressed SID endpoint, SRv6.SL=0 does not necessarily force an End.D* function. For information, see Section 41.2.1.1, Endpoint for Compressed SID Formats.
- When the SRv6 packet is received with SRv6.SL=1, the endpoint node might be configured to also perform PSP functionality. See Section 41.2.1.2, Endpoint PSP Mode.

Any endpoint node that is not a decapsulation function is fully agnostic to the header after the SRv6. The layer is not parsed and is not identified (unknown layer).

The SRv6 endpoint processing is as follows:

1. In termination stages, the packet is identified as an endpoint packet according to the SID format.

   Identifying the packet as My-DIP is similar to the IPv6 tunnel termination flow. The packet is classified with a new InLIF, and its properties, although there is no header termination in that process.

   At the end of that process, the FWD layer is still pointing on the IPv6 header, thus load-balancing is processed like a regular IPv6 forwarding packet.
2. In the FWD stage, the packet is forwarded according to the VRF from the IPv6 tunnel termination and the next SID.

3. In PMF, the next SID is sent explicitly to the egress pipe on the UDH. For more details about the PMF application requirements, see Section 41.2.1.3.1, Endpoint PMF Application.
4. To achieve an End.X functionality, set the InLIF with cross-connect information. Cross-connect means that the forwarding information is a property of the InLIF and is not found in the FIB. Yet, the packet is still processed as if the forwarding type is the IPv6 layer. This is done with the following APIs.
   a. Add the cross-connect indication when creating the InLIF with `bcm_tunnel_terminator_create(unit, *info)` using the `BCM_TUNNEL_TERM_CROSS_CONNECT` flag.
   b. Add cross-connect information with the `bcm_vswitch_cross_connect_add(unit, *gports)` API.

41.2.1.1 Endpoint for Compressed SID Formats

For the SRv6 compressed SID format, the behavior might be different because a single SID contains multiple compressed SIDs, and each endpoint node handles a single compressed SID.

41.2.1.1.1 Endpoint uSID

The uSID is a property of the InLIF (In-Logical-Interface). The uSID endpoint handling is performed for an SRv6 packet (with or without an SRH header) for which IPv6.DIP matches the endpoint's My-IP and the InLIF is set with the uSID property.

To set the InLIF with the uSID property, add the `BCM_TUNNEL_TERM_MICRO_SEGMENT_ID` flag during LIF creation.

A uSID endpoint node is any node that is classified as My-DIP and satisfies either of the following criteria:

- Its next uSID is not NULL (that is, IPv6.DIP[SID1] != 0), as shown in the following figure.

  SID1 represents a location on the IPv6.DIP, DIP[80:64]. In that case, the updated IPv6.DIP is as shown in the following figure.

  In that case, SRv6.SL is not decremented.
- Its next uSID is null, but SRv6.SL is not 0.
  In that case, the incoming packet IPv6.DIP is as shown in the following figure.

  In that case, a new IPv6.DIP is taken from the SRv6 SID list, and SRv6.SL is decremented.
- If both SRv6.SL=0 and IPv6.DIP[80:64] is NULL, the packet is processed as a regular egress node packet.

41.2.1.1.2 Endpoint GSID Node

The GSID is a property of the InLIF (In-Logical-Interface). GSID endpoint handling is performed when an SRv6 packet for which the IPv6.DIP matches My-DIP, and the InLIF is set with the GSID property.

To set the InLIF with the GSID property, add the `BCM_TUNNEL_TERM_GENERALIZED_SEGMENT_ID` flag during LIF creation.

At a GSID endpoint, the following processing occurs:

- IPv6.DIP is updated as follows:
  - The new GSID (32b) is taken from the SID list according to two parameters:
    - SRH.SL
    - IPv6.DIP.SI
  - SI is cyclically reduced by 1 (if it is 0, it is set to 3)
- If the incoming SI is 0 and reduced to 3, the SRH.SL is decremented by 1.

To exit the GSID domain, an GSID endpoint should also be defined as an *End of Compression* (EOC). The EOC is also an InLIF property, which is set by the `BCM_TUNNEL_TERM_GENERALIZED_SEGMENT_ID_END_OF_COMPRESSION` flag.

When an InLIF is a GSID EOC, the packet will exit the GSID domain, regardless of the SI value (it does not have to be 0), and the packet will be processed as follows:

- Classic SRv6 Endpoint – If SRH.SL > 0
- Egress node – If SRH.SL=0

DNX devices support up to four Compressed GSIDs (up to SL=3), that is, 12 GSIDs overall with a single VLAN tag at the outer Ethernet header.

41.2.1.2 Endpoint PSP Mode

A special endpoint behavior might occur when the SRv6 packet is received with SL=1. This is called endpoint PSP (penultimate segment popping).

An endpoint PSP is an endpoint functionality that also terminates the SRv6 extension.

As the PSP node routes the packet according to the last SID in the SRv6 SID list, it might be unnecessary to keep the SRv6 extension with its SID list, and a PSP node can terminate it.

A PSP handling is set for an endpoint packet with SRH.SL=1 and one of the following three options is valid:

- PSP global mode is set
- PSP indication per incoming SID (My-DIP)
- PSP indication per next SID

In addition, to allow a PSP termination for any SRv6 packet (that is, with any number of SIDs in the SID list) an extended-PSP flow is also supported.

The extended PSP flow is a combination of the regular PSP identification (by any of the three options) and the number of SIDs on the packet.

The handling of extended-PSP is performed at the PMF application and includes a dedicated recycle header (RCH), which is different from the standard RCH, called SRv6_USP_PSP_RCH.

The identification of the extended PSP flow is done in the ingress parser and set on the SRv6 layer qualifier (layer records). It contains a single bit to indicate whether a PSP flow can be processed in a single pass.

For more information on SRv6_USP_PSP RCH, see Chapter 17, Standardized Recycle Header.

41.2.1.2.1 PSP Global Mode

In PSP global mode, as part of the initialization of the application, it is required to configure whether the device supports endpoint PSP.

Use the `bcm_switch_control_set(unit, bcmSwitchSRV6EgressPSPEnable, 1)` API to configure the endpoint PSP support mode. In that mode, all endpoint packets with SRH.SL=1 are handled with PSP.

NOTE: If using the DNX application reference code as-is, you can also make sure that the API is being called by setting the SOC property:

```
appl_param_srv6_psp_enable=1
```

41.2.1.2.2 PSP Indication from Incoming DIP

PSP handling can be an InLIF property. The property is set on the InLIF with the following API:

```
bcm_port_class_set(unit, In-LIF-Gport, bcmPortClassFieldIngressVport,
<classification_value>)
```

The classification value in this API is any value that is used in the PMF to identify the flow. The range of the value can be configured as part of the PMF InLIF profile configuration (see Section 7.5, LIF Profile Resources).

See Section 41.2.1.3.1, Endpoint PMF Application for more details.

41.2.1.2.3 PSP Indication from Next SID

PSP handling per next SID is performed in the PMF application. It is identified with a dedicated TCAM lookup on the next SID in iPMF1.

See Section 41.2.1.3.1, Endpoint PMF Application for more details.

41.2.1.3 Configuration Checklist

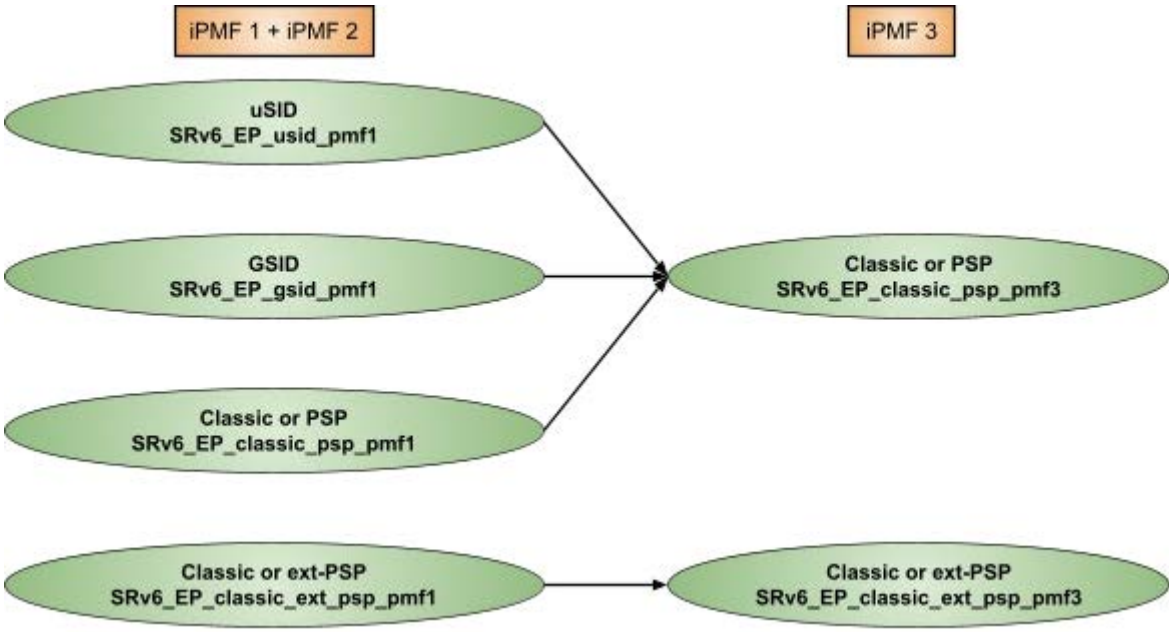
- Regular IP-Tunnel IPv6 termination object (see Chapter 30, IP Tunnel v6 Termination).
- Regular IP-Routing configuration.
- Field Processor Application Configuration for SRv6 endpoint node, which is part of the SDK Reference Application. See details in the following sections.

41.2.1.3.1 Endpoint PMF Application

This section describes the PMF application required for endpoint, endpoint PSP, and endpoint extended-PSP.

The following chart describes the contexts in each PMF stage, and it is followed with a description of the actions performed in each context.

Figure 33: PMF Stage Contexts



Context	Description	Selection
SRv6_EP_usid_pmf1	Builds UDH with the next IPv6.DIP according to the information in Section 41.2.1.1.1, Endpoint uSID	Selected according to the FWD2 context for SRv6 uSID forwarding.
SRv6_EP_gsid_pmf1	Builds UDH with the next IPv6.DIP according to the information in Section 41.2.1.1.2, Endpoint GSID Node	Selected according to the FWD2 context for SRv6 gSID forwarding.
SRv6_EP_classic_psp_pmf1	Builds UDH with the next SID. Identifies PSP cases according to the three options: ■ Global mode ■ TCAM lookup with next SID ■ TCAM lookup with the classification value (from InLIF) Sets a single bit to indicate PSP is used in PMF3.	Selected according to the forwarding flow, which is identified as an SRv6 endpoint, and SRH.SL is not 0.
SRv6_ext_psp_pmf1	Builds UDH with the next SID. Identifies PSP cases according to the three options: ■ Global mode ■ TCAM lookup with next SID ■ TCAM lookup with the classification value (from InLIF) Sets a single bit to indicate PSP is used in PMF3.	Selected according to the forwarding flow, which is identified as an SRv6 endpoint, and SRH.SL is 1 (for PSP eligibility). An additional pre-selector is that the number of SIDs are not eligible for PSP single pass.

Context	Description	Selection
SRv6_ext_psp_to_ep_pmf2	This context is present only if no PSP Global mode exists. The context checks if a *long packet* with SL=1 is actually an endpoint and not PSP. This context leads to SRv6_EP_classic_psp_pmf3 because the packet is treated as a regular endpoint packet.	Triggers after passing context in PMF1 SRv6_ext_psp_pmf1 (see the preselection information described previously), and preselect if PSP eligible cases have *not* been met by MyDIP or by next SID in PMF1.
SRv6_EP_classic_psp_pmf3	If the PSP indication is not set: ■ Update the parsing-start-type to the endpoint. ■ Update the parsing start offset to point to the IPv6 layer. If PSP indication is set: ■ Update the parsing-start-type to the endpoint PSP. ■ Configure the TM compensation to indicate the egress pipe will cut all SIDs from the packet (up to four in a single pass). ■ Update the parsing start offset to point to the IPv6 layer.	Cascaded from the previous contexts: ■ SRv6_EP_classic_psp_pmf1 ■ SRv6_ext_psp_to_ep_pmf2 ■ SRv6_EP_usid_pmf1 ■ SRv6_EP_gsid_pmf1
SRv6_EP_classic_ext_psp_pmf3	This context does the following: ■ Updates the parsing-start-type to PSP-USP-Extended ■ Configures TM compensation to indicate the egress pipe will cut exactly three SIDs from the packet ■ Overrides the destination to a dedicated RCY port. ■ Overrides OutLIF0 to an RCH OutLIF ■ Overrides InLIF0 with ETH-RIF ■ Sets the parsing-start-type to SRv6_EbdPoint_PSP_ext	Cascaded from the context SRv6_ext_psp_pmf1 if the PSP indication is set.

41.2.1.4 Application Reference

41.2.1.4.1 Field Processor Configurations

The Field Processor goal is to add additional functionality to the device to support the endpoint SRv6 packet.

Pass the next SID address for IPv6 layer editing (endpoint) and packet header indications, for correct parsing at Egress-pipe:

- **Type**: Application reference
- **Path**: `SDK/src/appl/reference/dnx/appl_ref_srv6_field_init_deinit.c`
- **Function**: `appl_dnx_field_srv6_init()`

41.2.1.4.2 Full Configuration

Basic and cross-connect endpoint for classic SID, uSID, and GSID and also used for PSP:

- **Type**: CINT reference
- **Path**: `SDK/src/examples/dnx/srv6/cint_srv6_basic.c`
- **Functions**:
  - `srv6_endpoint_tunnel()`
  - `srv6_endpoint_tunnel_cross_connect()`
  - `srv6_usid_endpoint_tunnel_cross_connect()`
  - `srv6_gsid_endpoint_tunnel_cross_connect()`

PSP extended configuration:

- **Type**: CINT reference
- **Path**: `SDK/src/examples/dnx/srv6/cint_srv6_basic.c`
- **Function**: `srv6_endpoint_psp_extended_configuration()`

PSP non-global mode configuration (by next SID, by MyDIP):

- **Type**: CINT reference
- **Path**: `SDK/src/examples/dnx/utility/cint_dnx_util_srv6.c`
- **Function**: `srv6_psp_tcam_configure()`

GSID full flow configuration:

- **Type**: CINT reference
- **Path**: `SDK/src/examples/dnx/tunnel/cint_dnx_srv6_gsid_full_flow_example.c`
- **Function**: `srv6_basic_gsid_example()`

41.2.2 Ingress Node

This section describes the following SRv6 functionalities:

H.Encaps, H.Encaps.Red, H.Encaps.L2, H.Encaps.L2.Red, End.B6.Encaps, End.B6.Encaps.Red H.Insert, H.Insert.Red, End.B6.Insert, End.B6.Insert.Red

The SRv6 ingress node is the node in the network that encapsulates the SRv6 tunnel based on the forwarding decision (OutLIF). Encapsulation of an SRv6 tunnel might include an encapsulation of an SRv6 and a IPv6 headers (Encaps functions), or an encapsulation of an SRv6 header only (Insert functions).

41.2.2.1 Reduced or Normal Encapsulation Mode

As part of the initialization of the application, it is required to configure whether the device works with Normal or Reduced mode.

In Reduced mode, on the ingress node of the SRv6 tunnel, the first SID that is used as the immediate address for the IPv6 header DIP is not included in the SRH header, thus saving 128b in the packet. In Normal mode, this SID is also included in the SRH.

The device mode (normal or reduced) is set by the following API:

```
bcm_switch_control_set(unit, bcmSwitchSRV6ReducedModeEnable, value)
```

NOTE: If using the DNX application reference code as-is, you can also make sure that the API is being called by setting the SOC property:

```
appl_param_srv6_reduce_enable=1
```

41.2.2.2 Encapsulation without an SRv6 Extension Header

For some encapsulation flows, building an SRH header is not required due to the number of SIDs that are encapsulated. This information applies to the following flows:

- H.Insert, H.Insert.Red with 0 SIDs – The packet looks like the IPv6 header was forwarded.
- END.B6.Insert, END.B6.Insert.Red with 0 SIDs – The packet looks like it was processed by a regular endpoint function.
- H.Encaps.Red with a single SID – The packet is encapsulated with an IPv6 header only.

41.2.2.3 SRv6 Encapsulation Objects

The SRv6 encapsulation process might contain an encapsulation of an IPv6 header, SRH, and SID list.

Each SID in the SID list is considered as 128b raw data, and the relevant API does not distinguish between a classic SID and compressed SID stack.

The SRv6 encapsulation includes four different types of OutLIFs:

- SRH OutLIF – `bcm_srv6_srh_base_initiator_create(unit, info)`
  - Holds the number of SIDs in the tunnel, which is taken into account when building the SRH fields Segment-Left, Last-Entry, and Extension-Length.
  - It should not take into account the VPN SID if the VPN SID was created with the IPv6-Tunnel OutLIF.
  - It has statistics, QoS, and trap capabilities.
- SID OutLIF – `bcm_srv6_sid_initiator_create(unit, info)`
  - 128b of raw data, no additional information.

- IPv6-Tunnel no DIP OutLIF – `bcm_tunnel_initiator_create(unit, intf, tunnel)`
  - An IPv6 tunnel OutLIF, created with Tunnel type:
    - `bcmTunnelTypeSR6`
    - `bcmTunnelTypeEthSR6`
  - The destination IP parameter (`dip6`) is 0.
- IPv6-Tunnel + VPN SID OutLIF – `bcm_tunnel_initiator_create(unit, intf, tunnel)`
  - An IPv6 tunnel OutLIF, created with Tunnel type:
    - `bcmTunnelTypeSR6`
    - `bcmTunnelTypeEthSR6`
  - The destination IP parameter (`dip6`) is set with the VPN SID (that is, not 0).
- IPv6-Tunnel + Part of VPN SID but no DIP OutLIF (High Scale VPN SID application) - bcm_tunnel_initiator_create(unit, intf, tunnel)
  - An IPv6 tunnel OutLIF, created with Tunnel type:
    - bcmTunnelTypeSR6
  - An IPv6 tunnel OutLIF, created with flags specially:
    - BCM_TUNNEL_INIT_SRV6_SCALE_VPN_SID

NOTE 1: VPN SID is a regular SID, and any flow that contains “IPv6-Tunnel + VPN SID OutLIF” can be replaced with a flow that contains “IPv6-Tunnel no DIP OutLIF” and “SID OutLIF”.

The reason to emphasize that an IPv6 header OutLIF can contain a VPN SID is to emphasize that in that case, the SID that is created with the IPv6 header object will be the last SID in the tunnel, which usually represents a specific service at the egress node.
VPN SID terminology is used in this chapter as it relates to the last SID in the tunnel.

41.2.2.4 Encapsulation Stack Structure

The SRv6 SIDs can be separated into Transport SIDs and a VPN SID. Transport SIDs are the SIDs in the tunnel that represent the route of the packet in the network, while the VPN SID is the last SID in the tunnel, which represents the service at the egress node.

The VPN SID usually contains additional information, such as an L2VPN or L3VPN service, an EVPN service, and maybe more.

The Transport and the VPN objects must be independent of each other, which means that the network should support multiple routes to the same end-user-service (protection consideration), or in contrast, multiple end-user-services that are passed on the same route.

There are several valid EES structures, depending on the required encapsulation.

- For Encaps functionalities:
  - IPv6-Tunnel OutLIF (with or without VPN SID)
  - SRH OutLIF
  - SID OutLIFs
  - ARP and AC OutLIFs
  - Native AC OutLIF (for Encap.L2)
- For Insert functionalities:
  - SRH OutLIF
  - SID OutLIFs
  - ARP and AC OutLIFs

Like any other encapsulation flow, each OutLIF is assigned to a specific EES phase. Some of the objects have a dedicated phase, and some are controlled by the user. The following table summarize the phases per object.

Table 95: EES Phase per Object

Object	Phase	Details
IPv6-Tunnel OutLIF (main part)	1	Not configurable
IPv6-Tunnel OutLIF (VPN part)	6 to 7	User configurable
SRH OutLIF	2	Not configurable
SID OutLIF	3 to 7	User configuration
ARP and AC OutLIFs	7 to 8	User configuration

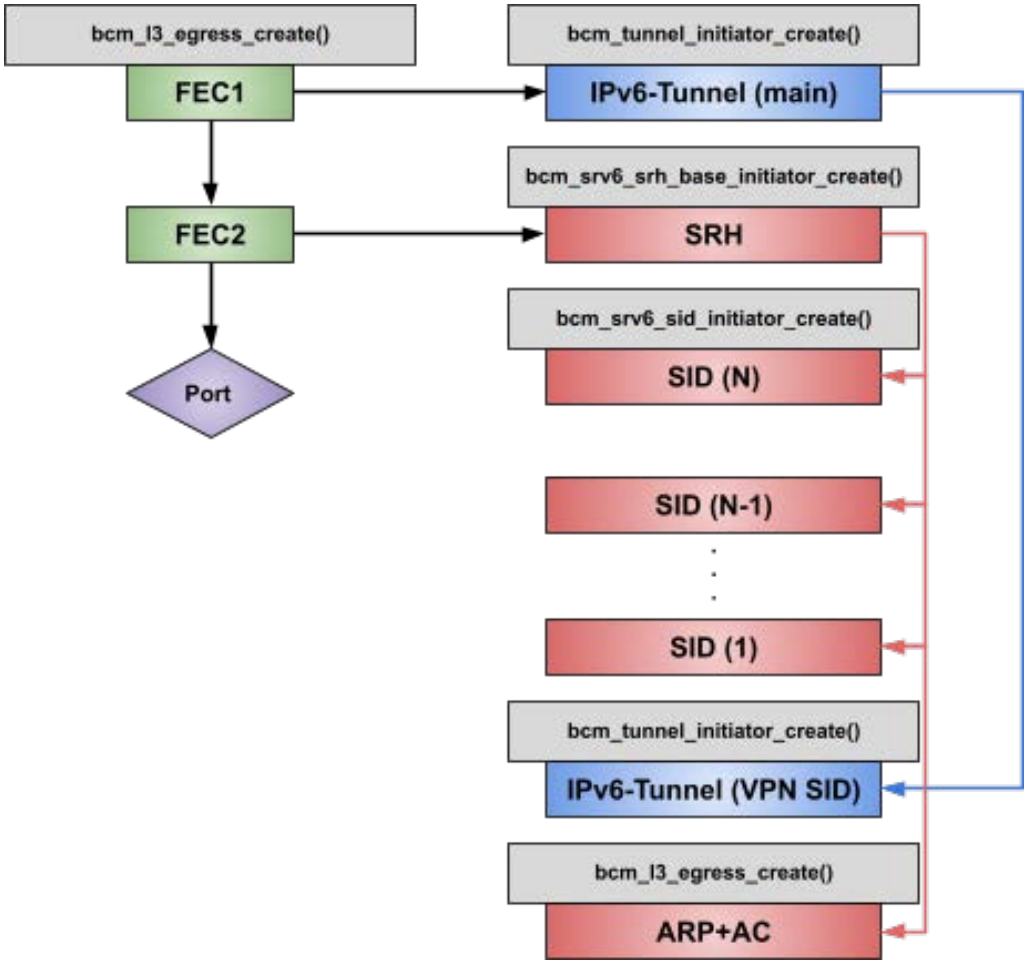
- ARP and AC – Can work in a single OutLIF or multiple OutLIFs (from EEDB or from ESEM).
- Native AC – From ESEM only because the IPv6 main part is fixed to phase 1.

41.2.2.5 Configuration Checklist

The following diagram suggests an SRv6 Ingress node configuration, from FEC to OutLIFs, and the received EES stack.

There are several ways to achieve a valid EES structure. OutLIFs can be linked one to the other in EEDB, or different OutLIFs can be pointed from different FEC hierarchies.

Figure 34: SRv6 Ingress Node Configuration



Users can decide on any other scheme that stands with the validity of the EES, for example:

- ARP can be pointed from a third-layer FEC.
- A single OutLIF from FEC, and concatenating all OutLIFs in the EEDB. In this option, the IPv6-Tunnel must be created without VPN SID because it should be chained with the SRH OutLIF.
- AC from ESEM
- ARP and AC are separated, and both are from the EEDB. In this case, the VPN SID should reside in phase 6. For details, see Section 41.4.4, IP-Tunnel IPv6 Encapsulation.
- No IPv6-Tunnel objects, for the Insert functionalities.

41.2.2.6 Extended Encapsulation

Extended encapsulation flows allow a larger number of SID encapsulation in a single node. Extended encapsulation flows are recycle flows that are based on recycle headers.

The RCY flow is configured using a dedicated RCY port and a new EES object, RCH-EE OutLIF. For more details about RCH and RCY flows. see Chapter 17, Standardized Recycle Header.

The API and the main parameters that the RCH-EE OutLIF holds are as follows:

- `bcm_l2_egress_create(unit, egr)`
- Next pass destination – Can be a FEC pointer or a port, using egr → dest_port
- A flag to indicate whether to use OutLIF0 from the previous pass in the next pass.

  `BCM_L2_EGRESS_EXTENDED_COPY_DEST_ENCAP`

  Set the flag on Encaps functionalities but not on Insert functionalities.

41.2.2.6.1 Encapsulation Stack Structure

A generic description of the extended encapsulation flows is separated into three types of passes: first, middle, and last pass. The following table that summarizes the processing and the ETPS structure in each pass.

Pass	Ingress Processing	Egress Processing	ETPS Structure
First	Standard processing of Incoming headers	■ Encapsulation of the inner SIDs ■ Consider reduce/normal mode ■ Encapsulation of RCH	1. 1. Native AC OutLIF (for Encaps.L2) 2. IPv6-Tunnel OutLIF (for Encaps) 3. SRH OutLIF 4. SID OutLIFs 5. RCH-EE OutLIF
Middle	None	■ Encapsulation of the middle SIDs ■ Ignore reduce/normal mode ■ Encapsulation of RCH	1. IPv6-Tunnel OutLIF (for Encaps) 2. SRH OutLIF 3. SID OutLIFs 4. RCH-EE OutLIF
Last	None	■ Encapsulation of the outer SIDs ■ Ignore reduce/normal mode ■ Encapsulation of SRH and IPv6 header ■ Standard processing of outer headers (ARP/AC)	1. IPv6-Tunnel OutLIF (for Encaps) 2. SRH OutLIF 3. SID OutLIFs 4. VPN SID 5. ARP and AC OutLIFs

NOTE:

- The extended object is always the Transport SIDs. The IPv6-Tunnel (and the VPN SID) are reused along the multiple passes.
- The IPv6 Tunnel OutLIF exists in all passes, although it is used (encapsulated) only in the last pass. The reason for that is that the IPv6-Tunnel OutLIF is independent of the transport layer and its length, so it must be found as the standard ingress processing, where the packet is sent with its OutLIF information.
- Each pass contains an SRH OutLIF, although the SRH is built in the last pass only. This OutLIF identifies the flow.

The number of SIDs stamped on the SRH is the number of SIDs set on the last pass SRH. Thus, it contains the total number of Transport SIDs in the tunnel.

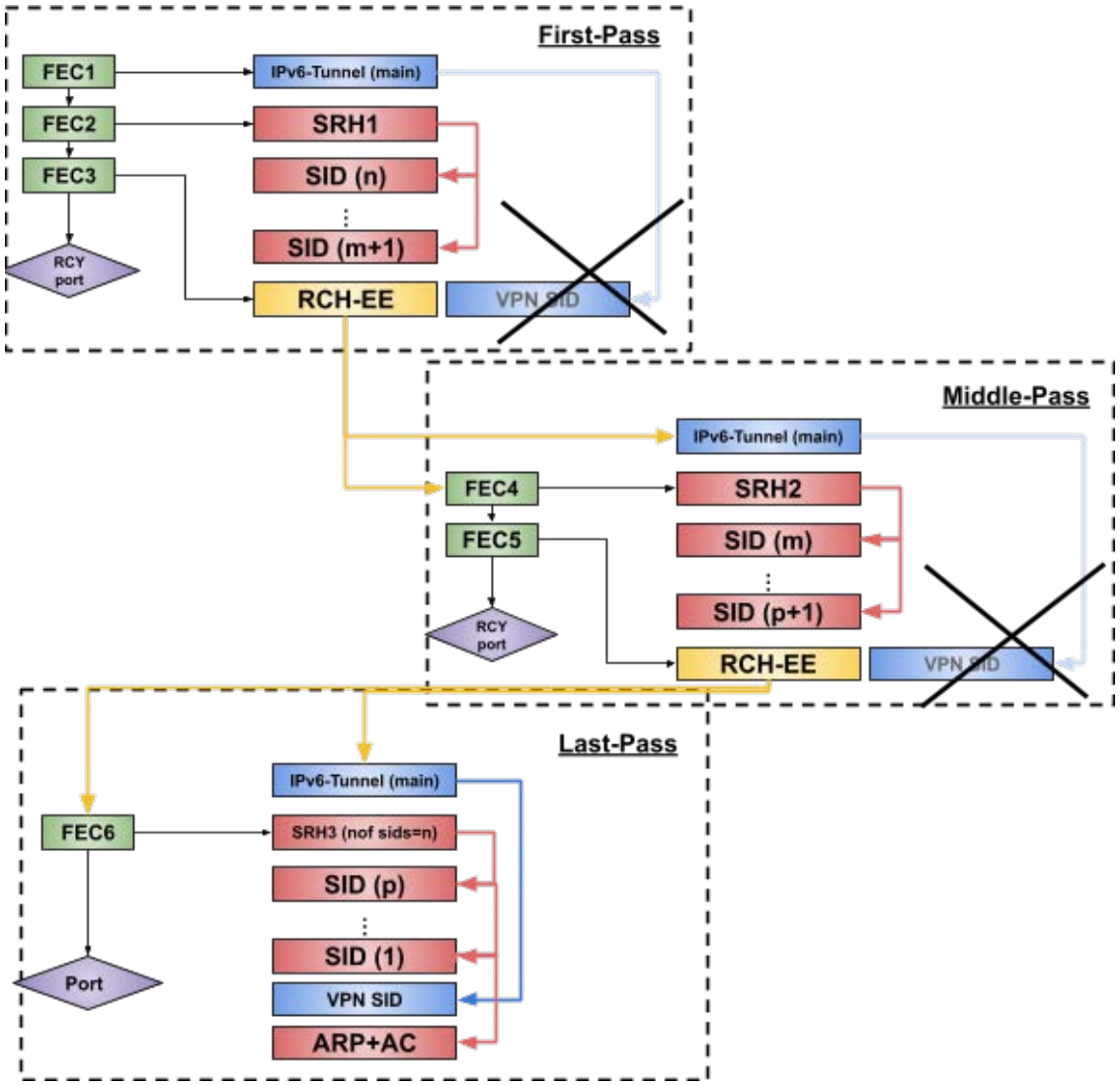
- A suggested configuration is to have three independent OutLIFs from ingress.
  - The VPN SID chained with the IPv6-Tunnel (EEDB link list)
  - The SRH and transport SIDs chained using EEDB link list
  - The RCH OutLIF

  The VPN SID appears in the EES only in the last pass. More details are in the following section.
- The RCH-EE OutLIF Phase is 7, which is the same phase at the VPN SID.

41.2.2.6.2 Configuration Checklist

The following diagram presents a suggestion of an extended encapsulation configuration from FEC to OutLIFs, and the received EES stack, assuming a three-pass process. The middle pass can be ignored in a two-pass process. (To simplify the diagram, the APIs are not added.)

Figure 35: SRv6 Ingress Node Configuration



NOTE:

- The yellow arrows in the figure are for cross-pass connections.
- The next-pass destination (FEC) is set on the RCH. It is configured by the user when creating the RCH OutLIF.
- The current-pass OutLIF0 is always stamped on the RCH.
- Users can control, per RCH OutLIF, whether to reuse the OutLIF in the ES stack of the next pass.
- The VPN SID is shaded out in the first pass and middle pass. This is achieved by having both the VPN SID OutLIF and RCH-EE OutLIF on the same EES phase and by having the RCH-EE OutLIF pointed directly from FEC, and not by the EEDB link list.

41.2.2.6.3 RCY Port Configuration

RCY port general configurations are described in Section 2.1, Packet Processing Ports. The RCY port is configured using the following API:

```
bcm_port_control_set(unit, port, bcmPortControlRecycleApp,
bcmPortControlRecycleAppExtendedEncap)
```

For more information about the RCY application type configuration, see Chapter 17, Standardized Recycle Header.

41.2.2.7 Application Reference

H.Encaps, H.Encaps.Red, H.Encaps.L2, H.Encaps.L2.Red, End.B6.Encaps, and End.B6.Encaps.Red:

- **Type**: CINT reference
- **Path**: `SDK/src/examples/dnx/srv6/cint_srv6_basic.c`
- **Function**: `srv6_ingress_tunnel_config()`

Extended encapsulation:

- **Type**: CINT reference
- **Path**: `SDK/src/examples/dnx/srv6/cint_srv6_ext_encap_tunnel.c`
- **Function**: `srv6_ingress_tunnel_ext_encap_config()`

H.Insert, H.Insert.Red, End.B6.Insert, and End.B6.Insert.Red:

- **Type**: CINT reference
- **Path**: `SDK/src/examples/dnx/srv6/cint_srv6_basic.c`
- **Function**: `srv6_ingress_tunnel_config()`

High Scale VPN SID application:

- **Type**: CINT reference
- **Path**:
  - `SDK/src/examples/dnx/srv6/cint_dnx_srv6_scale_full_flow_example.c`
  - `SDK/src/examples/dnx/field/cint_field_srv6_scale.c`
- **Function**: `dnx_basic_example_srv6_full_flow(), cint_field_external_pmf_main()`

41.2.3 Egress Node

This section covers the following SRv6 functionalities:

End.DX6, End.DX4, End.DT6, End.DT4, End.DT46, End.DX2, End.DX2V, End.DT2U, End.DT2M, USP and USD flavors.

The SRv6 egress node is the node in the network that handles SRv6 packets with SRH.SL=0. At the egress node, the SRv6 extension is terminated.

DNX devices support both USD and USP flavors, based on the protocols above SRv6.

- USD termination – The layer above SRv6 is IPv4, IPv6, or Ethernet. In that termination flow, the outer IPv6 layer is also terminated along with the SRv6 extension, and the pipe should forward the packet according to the inner header.
- USP termination – The layer above SRv6 is UDP, TCP, ICMP, or an inner SRv6 extension. In that termination flow, the outer IPv6 layer is kept, and only the first SRv6 extension is terminated. This flow must work in a two-pass process, while the second pass processes the header with the IPv6 original header.

This flow must work in a two-pass process, while the second pass will process the header with the IPv6 original header.

Egress node processing, whether it is a USD or a USP flow, is identified like regular endpoint processing, but for packets with SRH.SL=0.

41.2.3.1 USD Flow

A SRv6 USD termination is a flow that is configured differently for each number of SIDs on the SRv6 header, or in other words, it is configured per SRH.LE (Last Entry) value.

The USD service is selected with an ingress PMF on the egress node with SRH.SL=0 (an egress node), and the upper layer protocol is ETH, IPv4, IPv6, or MPLS.

If the packet is eligible for a single-pass USD termination, the configuration contains an InLIF tunnel, like in the endpoint case, and a forwarding entry, based on the next layer information.

NOTE: The packet is single-pass USD eligible according to device capability and number of SIDs on the SRH list.

Processing the packet with a two-pass termination process involves additional configurations in both the pipe and the PMF:

- The packet is mapped to a dedicated RCY port and an RCH OutLIF. The mapping is done according to the number of SIDs in the packet, that is, the SRH.LE field.

  The RCY port can be the same port for all mappings, but the RCH OutLIF is a dedicated OutLIF per number of SIDs.

  The API for that mapping is `bcm_srv6_extension_terminator_add(unit, *info)`.
- The packet is classified with the same InLIF in the second pass. For that, add the `BCM_TUNNEL_TERM_EXTENDED_TERMINATION` flag.

  To keep the original tunnel properties, the packet is reclassified in the second pass with the same InLIF from the first pass.
- The RCH OutLIF is configured as an extended-termination type. For more information, see Chapter 17, Standardized Recycle Header.
- The RCH OutLIF is mapped per number of SIDs because it holds parameters that decide the size of the header to terminate.
- Configure the RCY port as an extended-termination RCY port. For more details, see Chapter 17, Standardized Recycle Header.

In a two-pass USD process, the total number of bytes to terminate until exposing the next header is mapped from three places:

- PMF application – Set the termination of the outer ETH and additional 64B to cover IPv6, SRH, and the first SID.
- RCH OutLIF additional bytes to terminate – Additional bytes to add to first-pass termination.
- RCH OutLIF extension header size – The SRv6 header size that could not be terminated in the first pass, and considered as an RCH extension/padding, until the offset of the next layer.
- A configuration for the RCH OutLIF parameters with an assumption of system headers size of up to 34B is in the reference CINT `$SDK/src/examples/dnx/utility/cint_dnx_util_srv6.c`.

41.2.3.2 USP Flow

The SRv6 USP termination flow involves multiple passes. At least one pass must terminate the SRv6 extension, and an additional pass must process the IPv6 original header with the inner header.

After an endpoint node has SRH.SL=0 (an egress node), the USP process is selected by the upper layer protocol (that is not ETH, IPv4, IPv6, or MPLS), and the full configuration is performed in the PMF.

According to the number of SIDs on the SRv6 header, the process to terminate the SRv6 header might require multiple passes (extended-USP flow).

NOTE: A USP flow is always a multipass process. Assuming the process contains *N* passes (*N* ≥ 2), where:

- The *N*th pass will process the inner header.
- The *N*th – 1 pass will be the last pass of the SRv6 extension termination. This process is identical to the endpoint PSP process, where the only difference is that the UDH is built out of the original IPv6.DIP, and not the next SID.
- All previous passes are extended-USP processes, which are identical processes to the extended PSP flow.

In this document, the term *extended-USP* means the USP process requires at least two passes just for SRv6 extension termination and an additional pass to process the inner header.

A USP flow is identified in the pipe in the following two cases:

- Packet is SRv6oSRv6oIPv6oETH, where the outer SRv6 header has SL=0, and the IPv6.DIP is MyDIP (endpoint).
- The packet format is AnyoSRv6oIPv6oETH and the SRv6 header SL=0, and the packet destination (after FWDs) is a trap: bcmRxTrapSrv6Usp. This can be achieved with a simple route/host entry (KAPS/LEM) on the outer IPv6 header.

NOTE: As the route API `bcm_l3_route_add()` does not support setting the trap, an LPM result to have the trap set with an LPM lookup is required to configure the LPM result that should be translated to the USP trap.

To configure that LPM result, use `bcm_switch_control_set(unit, bcmSwitchControlRouteSrv6UspTrapSet, value)`, where `value` can be either a FEC (FORWARD_PORT gport), MCID (MCAST gport), raw value (no encoding), or –1, to disable the feature.

The application reference code is handled in both cases, so the packet should hit a dedicated USP PMF context, as shown below in the PMF diagram.

NOTE: In case of USP, not of SRv6oSRv6, but of TCP/UDP/ICMPoSRv6, the packet will be recycled with the same IPv6 header. If there was a routing entry to the USP trap, it will be routed into the USP trap also after removing the SRv6 extension.

The user is expected to handle that case in a dedicated PMF context, therefore, the packet will be routed to a CPU port. This is also handled by the application reference code.

41.2.3.3 Configuration Checklist

The egress-node tunnel flow contains the following items:

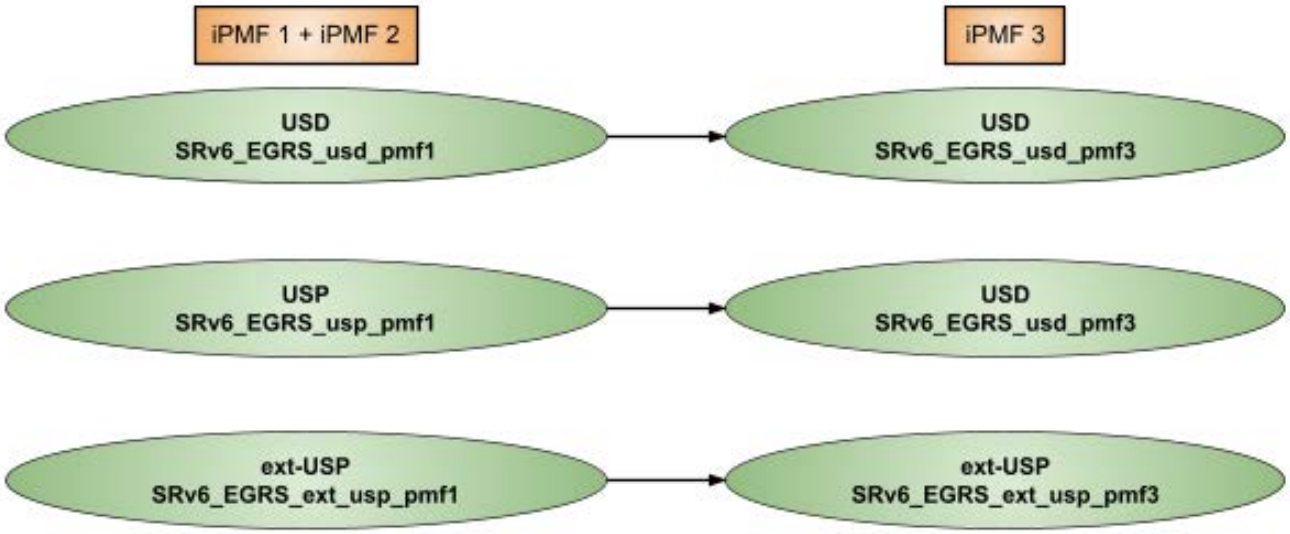
- Regular IP tunnel IPv6 termination object (see Chapter 30, IP Tunnel v6 Termination).
- Extended flow only: RCY port and RCY OutLIF configurations.
- Field processor application configuration for SRv6 egress nodes, which is part of the SDK reference application.

41.2.3.3.1 Egress Node PMF Application

This section describes the PMF application required for egress node, USP, and USD flavors.

The following chart describes the contexts in each PMF stage, and it is followed with a description of the actions done in each context.

Figure 36: PMF Stage Contexts



Context	Description	Selection
`SRv6_EGRS_usd_pmf1`	Used as a base context to be concatenated to the PMF3 context, as shown in Figure 36.	Selected by the SRv6 SL=0, and the upper layer is IPv4, IPv6, MPLS, or Ethernet.
`SRv6_EGRS_usp_pmf1`	Builds UDH with the IPv6.DIP to build the same IPv6 original header.	Selected by the SRv6 SL=0 with a lower priority than `SRv6_EGRS_usd_pmf1`.
`SRv6_EGRS_ext_usp_pmf1`	Used as a base context for iPMF3-relevant context.	Selected like the regular USP context but with a number of SIDs that are not eligible for single-pass termination processing.
`SRv6_EGRS_usd_pmf3`	Calculates teh parsing start offset to be ETH size + 64B. Updates the parsing start type with the relevant FWD header.	Selected with cascading from a iPMF1 context
`SRv6_EGRS_usp_pmf3`	The configuration is exactly like a single-pass endpoint PSP flow.	Selected with cascading from a iPMF1 context
`SRv6_EGRS_ext_usp_pmf3`	The configuration is exactly like a single-pass endpoint extended-PSP flow.	Selected with cascading from a iPMF1 context

41.2.3.4 Application Reference

41.2.3.4.1 Field Processor Configurations

The field processor adds additional functionality to the device to support the egress SRv6 packet.

Calculation of deep headers cutting (egress USD):

- **Type**: Application reference
- **Path**: `SDK/src/appl/reference/dnx/appl_ref_srv6_field_init_deinit.c`
- **Function**: `appl_dnx_field_srv6_init()`

41.2.3.4.2 Full Configuration

Regular Extended Termination (1)

- **Type**: CINT reference
- **Path**: `SDK/src/examples/dnx/srv6/cint_srv6_egress_tunnel_1_and_2_passes.c`
- **Function**: `srv6_egress_tunnel_basic()`

Regular Extended Termination (2)

- **Type**: CINT reference
- **Path**: `SDK/src/examples/dnx/tunnel/cint_dnx_srv6_tunnel_full_flow_example.c`
- **Function**: `dnx_basic_example_srv6_full_flow()`

RCY configuration and mapping

- **Type**: CINT reference
- **Path**: `SDK/src/examples/dnx/utility/cint_dnx_util_srv6.c`
- **Function**: `srv6_create_extended_termination_recycle_entry()`

Cross-connect

- **Type**: CINT reference
- **Path**: `SDK/src/examples/dnx/srv6/cint_srv6_basic.c`
- **Function**: `dnx_basic_example_srv6_full_flow()`

USP configuration, note that the extended termination of USP and PSP is similar; therefore, the reference is for a PSP extended CINT.

- **Type**: CINT reference
- **Path**: `SDK/src/examples/dnx/srv6/cint_srv6_basic.c`
- **Function**: `srv6_endpoint_psp_extended_configuration`
  The CINT adds configuration on top of context which were configured during application reference (appl-init).

High Scale VPN SID termination and validation

- **Type**: CINT reference
- **Path**:
  - `SDK/src/examples/dnx/srv6/cint_dnx_srv6_scale_full_flow_example.c`
  - `SDK/src/examples/dnx/field/cint_field_srv6_scale.c`
- **Function**: `dnx_basic_example_srv6_full_flow(), cint_field_external_pmf_main()`

41.2.4 SRv6 L2VPN

When the SRv6 tunnel is used as L2VPN (that is, an Ethernet header above SRv6), it may be configured in tagged mode or not. In case it is configured in tagged mode, the Inner-ETH (Native) In-AC (Native AC) can be classified based on the service delimiting VLAN Tags in the Inner-ETH. The number of service delimiting VLAN Tags in the packet is specified by the user or depends on the packet itself according to the user's applications.

Priority lookup is also supported as the PWE tagged mode in case it is a P2P case with flexible cross-connections. In this case, for example, if two service delimiting VLAN Tags are specified for the native AC classification, two lookups are done by two and one service delimiting VLAN Tags, separately. The latter lookup results are used if the former one fails to hit.

The functionality of SRv6 L2VPN is done with the following:

- Egress node:
  - Learning-enable and Learning-information can be performed only by the inner ETH In-AC.
  - The Inner-ETH (native) In-AC can be only used by the match criteria <Network-domain> classified with the Network-domain or the global tunnel inLIF. A Network-Domain lookup is performed by setting the next layer network-domain field from the SRv6 tunnel InLIF, which is configured with the `bcm_port_class_set(unit, In-LIF-GPort, bcmPortClassIngress, value)` API.
- Ingress node:
  - Native EVE – Native ETH VLAN editing is supported only by the VLAN-Port from ESEM, meaning it is supported by an exact match lookup triggered by the IPv6-Tunnel.
  - The lookup key for that case is: <OutLIF, Forward-Domain>.
  - The SRv6 encapsulation flow cannot encapsulate a routing-over-overlay (ROO) packet. To support this, the ROO encapsulation and SRv6 encapsulation must be split into a two-pass process.

NOTE: The indication of ETH over IP (that is, the IP *next protocol* field value) can be modified to identify Ethernet by using the `bcm_switch_control_set(unit, bcmSwitchIpv6NextProtocolEthernet, <59/143>)` API.

It can be toggled between 0/1. The default configuration, which corresponds to 1, is 143 (ETH, temporary value from RFC), otherwise, 0, which is corresponds to 59 (no next header).

41.2.4.1 SRv6 EVPN Termination

NOTE: The SRv6 EVPN feature is currently in beta version.

The SRv6 EVPN is a property of the InLIF tunnel interface of an SRv6 USD termination node.

When the next header above SRv6 is Ethernet, it is possible to control filtering of specific ports in a flooding packet flow. Filtered ports are signaled by an ESI label of 16b length in the packet. The incoming SRv6 packet contains the ESI label in the 16 LSBs of the IPv6 DIP.

EVPN functionality is supported for SRv6 packets with or without an SRH header, so it can also be applied on any IPv6 termination packet.

41.2.4.1.1 EVPN Termination

The configuration to support a USD EVPN node contains the following:

- Regular SRv6 egress USD node configuration, as in the SDK reference application.
- The IP tunnel terminator interface of the egress node must be classified as an EVPN LIF.

  The classification is set with the `bcm_port_class_set(unit, In-LIF-GPort, bcmPortClassFieldIngressVport, <classification value>)` API.
- Additional Ingress-PMF configuration that copies the ESI from the original packet into the UDH.
- Additional Egress-PMF configuration, which has similar functionality to the EVPN over MPLS functionality, described in Section 35.11, Field Processor Application for ESI Filtering, creating a database to control the filtering. If the ESI label is not allowed to be sent to the out-port, the packet will be dropped. Also, if the ESI label is not a known label (also configured in the Egress-PMF), the packet will be dropped.

For more information, see the Field Processor and CINTs application reference information in Chapter 35, Ethernet VPN (EVPN).

41.2.4.1.2 Application Reference

SRv6 L2VPN termination

- **Type**: CINT reference
- **Path**: `SDK/src/examples/dnx/tunnel/cint_dnx_srv6_tunnel_full_flow_example.c`
- **Function**: `dnx_basic_example_srv6_full_flow()`

SRv6 L2VPN termination in tagged mode

- **Type**: CINT reference
- **Path**: `SDK/src/examples/dnx/srv6l/cint_srv6_egress_tunnel_1_and_2_passes.c`
- **Function**: `srv6_egress_tunnel_end_dx2v_extended_p2p_main()`

EVPN

- **Type**: CINT reference
- **Path**: `SDK/src/examples/dnx/srv6/cint_dnx_srv6_evpn_basic.c`
- **Path**: `SDK/src/examples/dnx/field/cint_field_srv6_evpn_esi.c`
- **Functions**: All

41.3 SRv6 EVPN ESI Encapsulation

An ESI label identifies the original Ethernet segment, as in MPLS EVPN. It should be encapsulated at the bottom of the SID list (the last SID in forwarding) in the BUM traffic designated to the PE connected to the original Ethernet segment (ES), which is generally the designated forwarding PE. For more information about the ESI label, see Section 35.4, ESI Encapsulation.

The ESI label is determined by the IPv6 tunnel in the same way as the IML MPLS tunnel of MPLS EVPN. Create the IPv6 tunnel with a VPN SID, and define the 16 LSB of the VPN SID as ESI label, which can be updated per traffic destination.

41.3.1 Flexible ESI Solution

SRv6 EVPN includes a flexible solution by the egress PMF to retrieve the ESI label. To do this, define the preselectors and qualifiers to recognize the traffic that requires the ESI label, and set the action to update the IPv6 tunnel encapsulation pointer for the ESI encapsulation. The preselectors and qualifiers should include following keywords:

- A qualifier that indicates the BUM traffic if UC and MC share the same VPN SID
- A qualifier that indicates the source ES
- A qualifier that indicates the destination tunnel

For a physical ES mode, the source ES can be indicated by the source system port. For a virtual ES mode, the source ES can be indicated by the incoming AC interface.

The recommended egress-PMF action for updating the IPv6 tunnel encapsulation pointer is `bcmFieldActionAceEntryId`, which may be a multicast replication index (for multicast) or a global OutLIF (for unicast). If the IPv6 tunnel is encapsulated in the first pass, the action targets the multicast replication index. Otherwise, it targets the first global OutLIF.

41.3.2 Traditional ESI Solution

Another solution for the ESI encapsulation is to configure the ESI information into EM databases, which is the same solution that is used with MPLS EVPN. The IPv6 tunnel initiator entry triggers a lookup with the physical source port or the virtual source interface in the databases to retrieve the ESI label. A global default label is applied if nothing is found by the lookup. For more information about this solution, see Section 35.4, ESI Encapsulation.

41.3.3 Configuration Checklist

The configurations to support the flexible SRv6 EVPN ESI encapsulation include the following items:

- A regular SRv6 ingress node, as shown in the SDK reference application.
- An IP tunnel initiator with a VPN SID.
- An additional egress-PMF configuration to update the IPv6 tunnel encapsulation pointer with another pointer that has an ESI label.
- An additional egress-PMF or ingress-PMF configuration (or both configurations) to set the forwarding additional information field with the BUM traffic indicator, if necessary.

Additional configurations to support the traditional SRv6 EVPN ESI encapsulation include the following items:

- Create the IP tunnel initiator with a VPN SID and use one of the following flags to indicate it's an IML tunnel LIF:
  - `BCM_TUNNEL_INIT_IML` – IML tunnel LIF with a physical ESI.
  - `BCM_TUNNEL_INIT_IML` | `BCM_TUNNEL_INIT_ESI_INTF_NAMESPACE` – IML tunnel LIF with a virtual ESI.
- Add the ESI encapsulation information by calling the API with or without the virtual ESI indication flag, depending on the applications:

```
bcm_dnx_mpls_esi_encap_add( int unit, bcm_mpls_esi_info_t * esi_info)
```

41.3.4 Application Reference

Flexible SRv6 EVPN ESI encapsulation:

- **Type**: CINT reference
- **Path**:
  - `SDK/src/examples/dnx/field/cint_field_srv6_evpn_esi.c`
  - `SDK/src/examples/dnx/srv6/cint_dnx_srv6_evpn_basic.c`
  - `SDK/src/examples/dnx/srv6/cint_srv6_ext_encap_tunnel.c`
- **Function**: `srv6_evpn_esi_encap_main()`

Traditional SRv6 EVPN ESI encapsulation

- **Type**: CINT reference
- **Path**: `SDK/src/examples/dnx/srv6/cint_dnx_srv6_evpn_basic.c`
- **Function**: `srv6_evpn_mc_encap_1or2_pass_main(...)`

41.4 SRv6 APIs

41.4.1 SRv6 Extension Terminator

When configuring an SRv6 termination in two passes (extended-termination), map the packet to an RCH OutLIF and an RCY port. The RCY OutLIF contains some deep header cutting information, so the mapping should be done per number of SIDs on the SRv6 header.

41.4.1.1 SOC Properties

None.

41.4.1.2 Configuration Flow

`bcm_srv6_extension_terminator_add(unit, *info)`

- `info.flags` – `BCM_SRV6_EXTENSION_TERMINATOR_XXX` type flags.

Flags	Description
`BCM_SRV6_EXTENSION_TERMINATOR_REPLACE`	If set, the existing mapping is replaced.
`BCM_SRV6_EXTENSION_TERMINATOR_ADD_8B_EXTENSION`	If set, the extension header that is mapped is based on `nof_sids` (multiple of 16B) + additional 8B. This is used for 8B TLV packets.

- `info.nof_sids` – Number of SIDs on the SRv6 header
- `info.encap_id` – RCH OutLIF
- `info.port` – RCY port

41.4.2 SRH-Base Encapsulation

The SRv6 encapsulation consists of an SRH-base header and an SID list (explained in Section 41.4.3, SID Encapsulation). The SRH-base object allows creating an SRH-base header, which includes the information regarding the SRv6 tunnel, such as the number of segments and segments left to traverse.

The SRH-base object also stores Tunnel-RIF information, such as QoS parameters, MTU, and so on. It can point to SID encapsulation or an IPv6 tunnel. These parameters are shown in the following sections.

The SRH-base object ID is used as a Tunnel-Pointer by the ingress forwarding databases (such as the FEC database). For how to create an FEC and point to SRH-base objects (like any other Tunnel-pointer), see Section 22.8, L3 Egress Object: Ingress-FEC.

41.4.2.1 SOC Properties

If using the DNX application reference code, the `appl_param_srv6_reduce_enable` SOC property configures the application to enable reduced mode in all SRv6 encapsulations.

41.4.2.2 Configuration Flow

`bcm_srv6_srh_base_initiator_create(unit, *info)`

- `info.flags` – `BCM_SRV6_SRH_BASE_INITIATOR_XXX` type flags

Flags	Description
`BCM_SRV6_SRH_BASE_INITIATOR_WITH_ID`	If set, `tunnel_id` is provided by the user.
`BCM_SRV6_SRH_BASE_INITIATOR_REPLACE`	If set, updates the SRH base entry.
`BCM_SRV6_SRH_BASE_INITIATOR_STAT_ENABLE`	If set, indicates statistics is enabled.

- `info.tunnel_id` – Tunnel SRv6 SRH base object ID.
- `info.nof_sids` – Number of SIDs in the SID list.
- `info.egress_qos_model` – Egress QOS and TTL model. For more information, see Chapter 10, QoS.
- `info.ttl` – Tunnel header TTL. Used in case of Pipe models.
- `info.dscp` – Tunnel header DSCP value. Used in case of Pipe models.
- `info.qos_map_id` – QOS map identifier.
- `info.next_encap_id` – Next pointer interface ID to link to an additional encapsulation entry. Can be also set to invalid.
- To set the MTU value for an SRH-base object, call `bcm_rx_mtu_set(unit, mtu_config)`.
- `mtu_config.flags` = `MTU_RX_MTU_LIF`.
- `mtu_config.gport` = SRH-base object as a gport encoded.

NOTE: The SRH-base object is received encoded as intf. Users must apply the `BCM_L3_ITF_LIF_TO_GPORT_TUNNEL` macro to make it gport encoded.

- For the other parameters, see Chapter 12, Traps.

Statistics:

- `bcm_gport_stat_set(unit, gport, core_id, engine_source, stat_info)` where gport is the SRH-base object ID. For more information about the API, see Chapter 15, PP Statistics Generation.

NOTE: The SRH-base object supports stat-ID by default.

- For split-horizon filtering in L2VPN flows, the SRH base entry can be set with a classification group, set by the API:
  `bcm_port_class_set(unit, SRH-Base-Gport, bcmPortClassForwardEgress, <classification value>)`
- Overwrite the forwarding or mirroring ETPP trap action by calling:
  `bcm_rx_trap_lif_set(unit, flags, &lif_config)`
  – `lif_config.lif_gport` – SRH-base object gport ID.
  For more information, see Section 12.5.6, LIF Traps.

41.4.3 SID Encapsulation

The SRv6 encapsulation configuration consists of an SRH-Base header (discussed in Section 41.4.2, SRH-Base Encapsulation) and the SID list. The SID list provides the segment addresses in the SRv6 tunnel that are to be visited in an orderly way. Each segment entry from the SID list is represented as a different object and must be created. Its location in the SID list is determined according to its location (`encap_access`) in the encapsulation database.

NOTE: The SID encapsulation API is fully agnostic to the SID format.

The SID object also stores additional parameters, such as the linkage to another encapsulation entry.

41.4.3.1 SOC Properties

None

41.4.3.2 Configuration Flow

`bcm_srv6_sid_initiator_create(unit, *info)`

- `info.flags` = `BCM_SRV6_SID_INITIATOR_XXX` type flags.

Flags	Description
`BCM_SRV6_SID_INITIATOR_WITH_ID`	If set, `tunnel_id` is provided by the user.
`BCM_SRV6_SID_INITIATOR_REPLACE`	If set, updates the SID entry.
`BCM_SRV6_SID_INITIATOR_VIRTUAL_EGRESS_POINTED`	If set, creates the SID entry without a Global-LIF and pointed to by EEDB LL only. **NOTE:**

- `info.tunnel_id` – Tunnel SRv6 SID initiator object ID
- `info.sid` – SID to encapsulate
- `info.encap_access` – Specify the logical phase of the SID object. The value can be `bcmEncapAccessTunnel[1..4]` or `bcmEncapAccessArp` (if the VPN SID comes as Transport).

41.4.4 IP-Tunnel IPv6 Encapsulation

For information about how the IPv6 tunnel is created, see Chapter 31, IP Tunnel v6 Encapsulation. However, the IPv6 tunnel encapsulation usage in this section is different from the regular case of IPv6 tunnel used in IPv6 routing in that it serves the SRv6 tunnel. This means a few specific qualities are present when configuring the IPv6 tunnel encapsulation in the SRv6 application.

41.4.4.1 SOC Properties

None

41.4.4.2 Configuration Flow

`bcm_tunnel_initiator_create(unit, *intf, *tunnel)`

- `tunnel.type` – The tunnel type value and its details are in the following table.

Tunnel Types	Description
`bcmTunnelTypeSR6`	Indicates that the IPv6 tunnel might be encapsulated with an SRH-Base object. It applies the IPv6 tunnel to phase 1 and the DIP object to phase 7.
`bcmTunnelTypeEthSR6`	The same as `bcmTunnelTypeSR6`, but also eligible for ESEM lookup to retrieve Native AC information.

- `tunnel.dip6` – SID VPN.
- `tunnel.sip6` – Configure tunnel SIP to be encapsulated.

- `tunnel.encap_access` – Controls only the VPN SID location in the ETPS stack. If the default value is used, the main entry is in phase 1, and the VPN SID entry is in phase 7. If `bcmEncapAccessTunnel4` is used, the VPN SID is moved to phase 6.

All other parameters are similar to the regular IPv6 tunnel encapsulation, as described in Chapter 31, IP Tunnel v6 Encapsulation.

41.4.5 IP-Tunnel IPv6 Termination

The IPv6 tunnel is the underlying infrastructure of the SRv6 tunnel, so it also undergoes termination at endpoint and egress nodes. For information about how the IPv6 termination occurs, see Chapter 30, IP Tunnel v6 Termination.

41.4.5.1 Tunnel Scale Optimization

My-DIP identification uses standard-IP tunnel identification but with only TCAM to identify the entire 128-bit DIP. This process might be not scalable because the number of the SRv6 tunnels might be larger than the TCAM scale.

For that reason, there is an option to optimize the My-DIP lookup process so only the locator is searched at the TCAM, and the function (which is the large-scale element) is held in an EM lookup.

The optimization options are as follows:

- Classic SID: Locator 96b, function 16b
- Classic SID: Locator 64b, function 16b
- uSID: Locator 48b (prefix + uSID), no function
- GSID: Locator 80b (prefix 48b + GSID), no function

When using one of the optimized flows, the tunnel termination flow is as follows:

- Lookup in TCAM:
  - The key is the locator bits
  - The result is as follows:
    - Locator LIF (`default_lif`).
    - Locator type
- Lookup in ISEM:
  - The key is the Locator LIF and the function bits
  - The result is a LIF.

NOTE:

- If the Locator-LIF is found and the function-LIF is not found, the locator-LIF is used as the InLIF.
- Any locator or function InLIFs allow tunnel termination processing (that is, My-DIP).

The configuration flow for that purpose contains two calls to the `bcm_tunnel_terminator_create(unit, &tunnel)` API.

- The first call defines the locator entry and the locator LIF, which requires one of the following flags that indicate the locator and function format:
  - `BCM_TUNNEL_TERM_UP_TO_96_LOCATOR_SEGMENT_ID`
  - `BCM_TUNNEL_TERM_UP_TO_64_LOCATOR_SEGMENT_ID`
  - `BCM_TUNNEL_TERM_MICRO_SEGMENT_ID`
  - `BCM_TUNNEL_TERM_GENERALIZED_SEGMENT_ID`
- The second call defines the function LIF, and it should contain the following inputs:

- `tunnel.type` – `bcmTunnelTypeCascadedFunct`
- `tunnel.default_tunnel_id` – The `tunnel_id` from the first call, the locator tunnel ID.

41.4.5.2 SOC Properties

None

41.4.5.3 Configuration Flow

`bcm_tunnel_terminator_create(unit, *tunnel)`

- `tunnel.type` – The tunnel type value and its details are in the following table.

Tunnel Types	Description
`bcmTunnelTypeIpAnyIn6`	Creates an InLIF that can be classified for packets of type SRv6oIPv6 or IPv6 with and without an SRv6 extension. InLIF can hold a VRF value.
`bcmTunnelTypeEthIn6`	Creates an InLIF that can be classified for packets of type SRv6oIPv6 or IPv6 with and without an SRv6 extension. InLIF must hold a valid VSI value.
`bcmTunnelTypeCascadedFunct`	Creates an InLIF that is classified in EM, and not in TCAM. When using that flag, it is required to provide a valid `default_tunnel_id`. For more information, see Section 41.4.5.1, Tunnel Scale Optimization.
`bcmTunnelTypeEthCascadedFunct`	Same as the previous type, but the InLIF must hold a valid VSI value or a cross-connect indication.

- `tunnel.flags` – The tunnel flags details are in the following table

Flags	Description
`BCM_TUNNEL_TERM_UP_TO_96_LOCATOR_SEGMENT_ID`	See Section 41.4.5.1, Tunnel Scale Optimization.
`BCM_TUNNEL_TERM_UP_TO_64_LOCATOR_SEGMENT_ID`	See Section 41.4.5.1, Tunnel Scale Optimization.
`BCM_TUNNEL_TERM_MICRO_SEGMENT_ID`	Set an InLIF with the uSID property.
`BCM_TUNNEL_TERM_GENERALIZED_SEGMENT_ID`	Set an InLIF with the GSID property.
`BCM_TUNNEL_TERM_GENERALIZED_SEGMENT_ID_END_OF_COMPRESSION`	Set an InLIF that has GSID property, as an End of compression GSID zone. These can be used with the flag `BCM_TUNNEL_TERM_GENERALIZED_SEGMENT_ID` or with the following tunnel types: ■ `bcmTunnelTypeCascadedFunct` ■ `bcmTunnelTypeEthCascadedFunct`
`BCM_TUNNEL_TERM_CROSS_CONNECT`	Specify the tunnel is used for cross-connect. **NOTE:** It is not possible to switch between cross-connect and multipoint tunnels, and vice versa.
`BCM_TUNNEL_TERM_EXTENDED_TERMINATION`	Specifies the tunnel is eligible for extended-termination flow. For extended termination-flow, the tunnel must be reclassified in the second pass.
`BCM_TUNNEL_TERM_SERVICE_TAGGED`	Specify the L2VPN tunnel is working in tagged mode. If it is set, either specify the service delimiting VLAN Tags by `tunnel->nof_service_tags` or by the packet format.

Flags	Description
BCM_TUNNEL_TERM_SERVICE_TAGGED_BASED_HEADER_ONLY	If the L2VPN tunnel is working in tagged mode, this flag classifies the native AC by the packet format itself rather than the given nof_service_tags.
BCM_TUNNEL_TERM_SERVICE_TAGGED_PLATFORM_NAMESPACE	Specify the native AC is classified in the Network-domain scope for the L2VPN tunnel. If it is not set, the native AC is classified in the tunnel InLIF scope.

- tunnel.dip6 – Configure the DIP value for lookup.
- tunnel.sip6 – Configure the SIP value for lookup.
- tunnel.vrf – Configure the VRF that is used to perform tunnel termination.
- tunnel.vlan – Configure the VSI that is updated from the tunnel termination. Can be used only with tunnel types bcmTunnelTypeEthIn6 and bcmTunnelTypeEthCascadedFunct.

NOTE: The value of this parameter is allowed to be 0 only in cases other than the specified cases or if the cross-connect flag is used.

- tunnel.ingress_qos_model – Ingress QoS and TTL model. For more information, see Chapter 10, QoS.
- tunnel.default_tunnel_id – SID scale optimization use case. Valid only with the bcmTunnelTypeCascadedFunct tunnel type. Set this with the locator LIFs.

NOTE: The cascaded tunnel functionality is possible for both L2 and L3VPN tunnels. The final forwarding domain type and value are resolved from the function LIF. It is not necessary to match the value of the locator.

- For a L2VPN function LIF tunnel, specify tunnel.vlan attribute. For L3VPN, the VRF is set with the bcm_l3_ingress_create API.

For other tunnel parameters, see Chapter 30, IP Tunnel v6 Termination.

41.5 API Descriptions

41.5.1 bcm_srv6_srh_base_initiator_*

API Name	Highlights
bcm_srv6_srh_base_initiator_create()	Create SRH base entry and a link to an additional entry
bcm_srv6_srh_base_initiator_get()	Retrieve SRH base entry details
bcm_srv6_srh_base_initiator_delete()	Delete SRH base entry
bcm_srv6_srh_base_initiator_traverse()	Call a user-defined callback over all SRH Base objects

41.5.2 bcm_srv6_sid_initiator_*

API Name	Highlights
bcm_srv6_sid_initiator_create()	Create SID entry and link to an additional entry
bcm_srv6_sid_initiator_get()	Retrieve SID entry details
bcm_srv6_sid_initiator_delete()	Delete SID entry
bcm_srv6_sid_initiator_traverse()	Call a user-defined callback over all SID objects

41.5.3 bcm_srv6_extension_terminator_*

API Name	Highlights
`bcm_srv6_extension_terminator_add()`	Retrieve details on mapping of number of SIDs in packet to an RCY port and encapsulation entry
`bcm_srv6_extension_terminator_get()`	Retrieve details on mapping of number of SIDs in packet to an RCY port and encapsulation entry
`bcm_srv6_extension_terminator_delete()`	Delete the mapping of number of SIDs to an RCY port and encapsulation entry
`bcm_srv6_extension_terminator_traverse()`	Call a user-defined callback over all number of SIDs mappings

41.6 SRv6 Functions Support List

The following SRv6 functionalities are supported and available:

- END, END.X, END.T
- END.DX2, END.DT2U, END.DT2M, END.DX4, END.DX6, END.DT4, END.DT6, END.DT46, H.Encaps, H.Encaps.RED,H.Encaps.L2, H.Encaps.L2.RED, END.B6.Encaps, and END.B6.Encaps.Red
- H.Insert, H.insert.RED, END.B6.Insert, and END.B6.Insert.Red
- All In USD, USP, and PSP flavors, with SIDs formats Classic, uSID, and GSID

Chapter 42: TWAMP

42.1 Introduction

Two-Way Active Measurement Protocol (TWAMP), specified in [RFC5357], is a protocol for measuring delay and packet loss on a link, based on One-way Active Measurement Protocol (OWAMP, [RFC4645]).

OWAMP provides a common protocol for measuring one-way metrics between network devices. It may be used bi-directionally to measure one-way metrics in both directions between two network elements. However, it does not accommodate round-trip or two-way measurements.

TWAMP adds two-way or round-trip measurement capabilities. The TWAMP measurement architecture is usually comprised of two hosts with specific roles, which allow for protocol simplifications.

In TWAMP, a network device may be used, within a session, as either a Session-Sender or a Session-Reflector.

TWAMP is an open protocol for measurement of two-way metrics, based on OWAMP's overall architecture and design.

TWAMP defines two protocols:

- TWAMP-Control:
  - The Control-Client sets up test sessions by initiating a TCP connection to a server, where the two negotiate and agree on test modes and settings.
  - The Control-Client creates sessions with roles of Session-Sender (normally the Control-Client itself) and Session-Reflector (normally the Server).
  - The Control-Client orders the Server to start/stop/end a session.
- TWAMP-Test

  Packets are sent by the Session-Sender. For every packet received by the Session-Reflector, it responds immediately with a packet containing the parameters of the received packet, with additional parameters, back to the Session-Sender.

Each of the two protocols has known roles. TWAMP-Control is exchanged between the Control-Client and the Server. TWAMP-Test is exchanged between Session-Sender and Session-Reflector

42.1.1 TWAMP-Control Protocol

TWAMP-Control is used for creating the initial connection between the parties, initiating sessions, negotiating their parameters, and controlling their start/stop. The protocol is managed by the Control Client.

The roles of the Control-Client are:

- Sets up the connection:
  - Initiates a TCP connection with the TWAMP Server well-known port (862).

    Session roles are now defined. The TCP initiator is the Control-Client/Session-Sender; the acknowledging host becomes the Server/Session-Reflector.

    DSCP from the IP header of the TCP SYN packet should be recorded and used in all packets related to the session.
  - Queries the server for supported modes, unauthenticated/authenticated/encrypted.
  - Chooses modes and notifies the Server.
- Creates sessions:
  - Sends Request-TW-Session command and waits for Accept-Session message from Server
- Orders the start/stop of using Start-Sessions and Stop-Sessions messages.

The roles of the Server are:

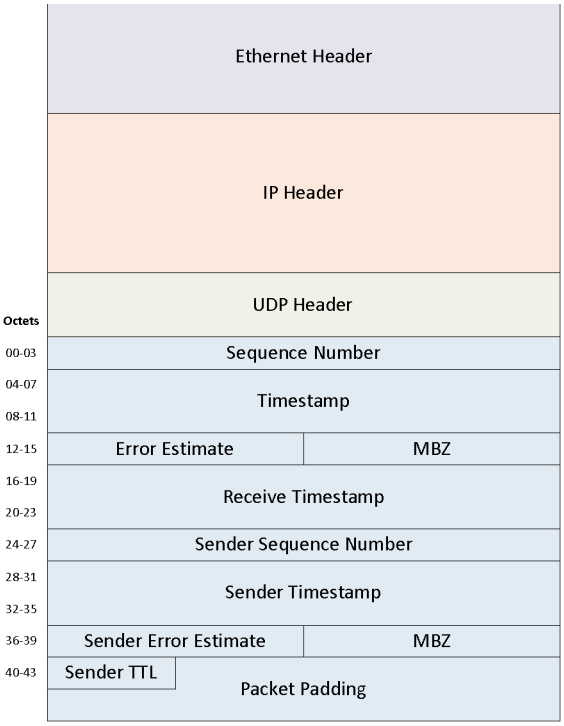
- Participates in setting up the connection:
  - Acknowledges TCP SYN message.
  - Stores the DSCP from the IP header of the TCP SYN packet to use in all packets related to the session.
  - Responds to Client query on supported modes
  - Sends Server-Start message to complete connection setup procedure
- Session creation:
  - Responds to Request-TW-Session command with Accept-Session message.
- Responds to Start-Sessions message with Start-Ack.

42.1.2 TWAMP-Test

Once the session starts, the Control-Client assumes the role of the Session-Sender and the Server assumes the role of the Session-Reflector.

The Session-Sender starts transmitting packets. The packet's transmission schedule (traffic profile) is considered outside the scope of TWAMP standard. The structure of a TWAMP-Test Sender Packet is shown below.

Figure 37: TWAMP-Test Sender Packet



The packet encodes the following fields:

- Sequence Number – Start with zero and incremented by one for each subsequent packet
- Timestamp – Time of transmitting the packet
- Error Estimate – Estimate of the error of synchronization, in scale/multiplier format

  The Error Estimate specifies the estimate of the error and synchronization. It has the following format:

```
 0                   1
 0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|S|Z|   Scale   |   Multiplier  |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
```

  The first bit, S, *should* be set if the party generating the timestamp has a clock that is synchronized to UTC using an external source (that is, the bit should be set if GPS hardware is used and it indicates that it has acquired the current position and time or if NTP is used and it indicates that it has synchronized to an external source, which includes stratum 0 source, and so on). If there is no notion of external synchronization for the time source, the bit *should not* be set. The next bit has the same semantics as MBZ fields elsewhere. It *must* be set to zero by the sender and ignored by everyone else. The next six bits, Scale, form an unsigned integer; Multiplier is an unsigned integer as well. They are interpreted

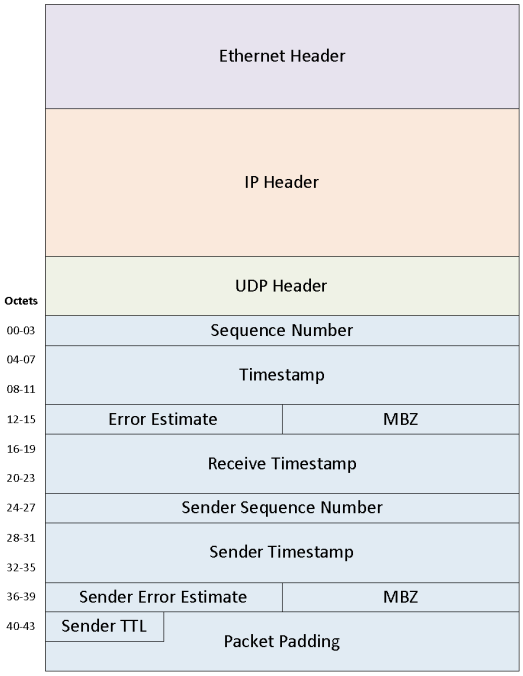
as follows: The error estimate is equal to Multiplier × $2^{(-32)}$ × 2^{Scale} (in seconds). Notation clarification: 2^{Scale} is two to the power of Scale.The multiplier *must not* be set to zero. If the multiplier is zero, the packet *should* be considered corrupt and discarded.

- Packet Padding – All bytes beyond the first 14. The length of the padding is defined in the Session-Request message. Padding may either be pseudo-random or zeros, per configuration.

In addition, the TTL in the packet's IP header is be set by the sender to 255.

For every Sender Packet received at the Reflector, the Reflector immediately responds with a TWAMP-Test Reflector Packet, shown below.

Figure 38: TWAMP-Test Reflector Packet



The Reflector packet copies some fields from the received sender packet and generates other fields, as follows:

- Sequence Number – Starts with zero and is incremented by one for each subsequent packet. This field is generated by the Reflector, independently of the Sender Sequence Number.
- Timestamp – Time of transmitting the packet.
- Error Estimate – An estimate of the error of synchronization, in scale/multiplier format.
- Receive Timestamp – The time the packet was received in the Reflector.
- Sender Sequence Number – Copied from the packet received from the sender (TWAMP Sequence number field).
- Sender Timestamp – Copied from the packet received from the sender (TWAMP Timestamp field).
- Sender TTL – Copied from the packet received from the sender (IP header TTL)
- Packet Padding – All bytes beyond the first 42. The length of the padding is defined in the Session-Request message. Padding may either be pseudo-random or zeros, per configuration.

In addition, the TTL in the packet's IP header is set by the Sender to 255.

The test is completed when the Control-Client/Sender stops transmitting packets or when it transmits a Stop-Sessions command.

42.1.3 TWAMP over LAG

The TWAMP over LAG functionality is based on the *Performance Measurement on LAG* IETF draft (https://tools.ietf.org/html/draft-li-ippm-pm-on-lag-03).

TWAMP sessions run between individual LAG members on two directly connected devices (similar to micro-BFD). For TWAMP over LAG, additional fields in the TWAMP header identify the member IDs of the sender and the reflector. For regular TWAMP, the sender member link ID and reflector member link ID are filled with MBZ (0).

The packet encodes the following extra fields compared with the regular TWAMP Sender Packet:

- Sender Member Link ID – Carries the LAG member link identifier of the Sender side. The value of the Sender Member Link ID *must* be unique at the Session-Sender.
- Reflector Member Link ID – Carries the LAG member link identifier of the Reflector side. The value of the Reflector Member ID *must* be unique at the Session-Reflector.

The packet encodes the following extra fields compare with the regular TWAMP Reflector Packet:

- Sender Member Link ID – Carries the LAG member link identifier of the Sender side. The value of the Sender Member Link ID *must* be unique at the Session-Sender.
- Reflector Member Link ID – Carries the LAG member link identifier of the Reflector side. The value of the Reflector Member ID *must* be unique at the Session-Reflector.

42.1.4 UDP Checksum

42.1.4.1 TWAMP Reflector

The TWAMP sender packet must contain the correct UDP checksum. The TWAMP reflector packet UDP checksum calculation is based on the received packet UDP checksum.

- TWAMP over IPv4 – TWAMP sender packet padding can be pseudo-random or zeros.
- TWAMP over IPv6 – TWAMP sender packet padding must be all zeros.

42.1.4.2 TWAMP TX (Non-Accelerated and Accelerated)

The TWAMP TX sender packet must contain the correct UDP checksum, but the UDP checksum should not calculate the sequence number. The TWAMP TX packet UDP checksum calculation is based on the received packet UDP checksum.

42.2 Application Configuration Checklist

42.2.1 TWAMP Reflector Configuration

- The packet is received from the network through a front-panel port.
- The ingress PMF detects the packet by its header fields and attaches an appropriate CPU Trap Code (for recycle), OutLIF indicating TWAMP-Reflector Swap operation, and OAM header with RX timestamp.
- The ETPP swaps SIP/DIP and UDP Src/Dst ports, stamps TTL=255, copies received packet fields to the appropriate location in the transmitted packet, and stamps RX timestamp in the appropriate location.
- The ETPP also stamps the Sender Member Link ID from the transmitted packet and the Reflector Member Link ID from the ETPS entry for the TWAMP-over-LAG.
- The packet is recycled for another trip through ingress.
- The IRPP processes the packet as an IP packet and forwards it according to the IP address.

- The ingress PMF detects the packet by its header fields and adds an OAM header with a counter ID and instructions for the ETPP to stamp the TX timestamp, sequence number, and to not decrement TTL.
- The ITM accesses the CRPS to replace the counter ID with the counter value (sequence number).
- The ETPP stamps the TX Timestamp, sequence number, and encapsulates the packet as instructed by the Ingress.
- The UDP checksum is correct.
- The packet is transmitted using a front-panel port.

42.2.2 TWAMP TX Non-Accelerated Configuration

- The CPU constructs and injects packets with all the needed fields except for TX timestamp.
- The IRPP processes the packet as an IP packet and forwards it using the IP address.
- The ingress PMF detects the packet by its header fields and instructs the ETPP to stamp the TX timestamp within the TWAMP-Test packet, and to not decrement TTL.
- The ETPP stamps the TX Timestamp and encapsulates the packet as instructed by the Ingress.
- The packet is transmitted using a front-panel port.

42.2.3 TWAMP RX Non-Accelerated Configuration

- The packet is received from the network through a front-panel port.
- The ingress PMF detects the packet by its header fields and directs it to the CPU, with Trap Code, Trap Qualifier, and the RX timestamp in the system header.
- The egress pipe traps the packet, with its system headers, to the CPU.
- The CPU processes the packet.

42.2.4 TWAMP Accelerated TX

- Configure the accelerated GTF.
- The accelerator constructs and injects packets with all the required fields except for the TX timestamp.
- The IRPP processes the packet as an IP packet and forwards it by using the IP address.
- Ingress PMF detects the packet by its header fields and instructs the ETPP to stamp the TX timestamp within the TWAMP-Test packet, and to not decrement TTL.
- The ETPP stamps TX timestamp and encapsulates the packet as instructed by the ingress.
- The packet is transmitted through a front-panel port.

42.2.5 TWAMP Accelerated RX

- Configure the accelerated CTF.
- Clean the accelerated CTF statistics.
- The packet is received from the network through a front-panel port.
- The ingress PMF detects the packet by its header fields and directs it to the CPU, with Trap Code, Trap Qualifier, and the RX timestamp in the system header.
- The egress pipe traps the packet, with the system headers, to the accelerator.
- Collect the accelerated CTF statistics.

42.3 TWAMP Reflector

In this application, the device is configured to execute the role of the Session-Reflector. This feature is supported only on JR2 system headers.

42.3.1 TWAMP Reflector Mode

TWAMP reflector supports standard-track full TWAMP, which maintains test-session context (sequence in CRPS). It also supports STAMP/TWAMP-Light in stateless mode. It is configured by `bcm_switch_control_set(unit, bcmSwitchTwampStatelessModeEnable, value);`.

42.3.2 SOC Properties

Recycle port configuration (example):

- `ucode_port_40=RCY.0:core_0.40`
- `tm_port_header_type_in_40=RCH_0`
- `tm_port_header_type_out_40=ETH`

42.3.3 Configuration Flow

1. Configure routing (My-Mac, RIF, VRF; see Chapter 22, IP Router).
2. Call `bcm_l3_egress_create` to create FEC (see Chapter 22, IP Router).
3. Add IP route to FEC (see Chapter 22, IP Router).
4. Create RCH Ethernet encapsulation (see Chapter 18, Drop-and-Continue).
5. Create TWAMP encapsulation to RCH Ethernet encapsulation.
   a. Use `bcm_switch_reflector_create()`
      - `error_estimate`: Error estimate
      - `reflector_member_link_id`: Reflector member link ID for TWAMP LAG
      - `next_encap_id`: RCH Ethernet encapsulation pointer
6. Create a trap with the following (see Chapter 12, Traps):
   – Destination – Recycle port
   – `encap_id` – TWAMP encapsulation OutLIF
   – (Optional) `snoop_cmnd` – Snoop to Accelerator
7. Set recycle port to overlay packets in the second pass.
8. Create a CRPS for the SEQ value. Refer to the *Counter Processor* section in the *Traffic Manager* document.
9. Create the PMF for the first pass:
   a. Identify received TWAMP-Reflector packets.
   b. Redirect to the recycle port.
   c. Add RX timestamp in system header (Down-MEP).
   d. Add OutLIFs (TWAMP-REFLECTOR + RCH).
   e. Reset TTL to 255.
10. Create the PMF for the second pass.

a. Identify received TWAMP-Reflector (second pass) packets
b. Update destination instructions for TX timestamping, SEQ stamping, and not decrementing TTL.

42.3.4 Shell Commands

None

42.3.5 Application Reference

CINT example with detailed description of the application.

- **Type:** CINT reference
- **Paths:**
  - `$SDK/src/examples/dnx/twamp/cint_dnx_twamp_field.c`
  - `$SDK/src/examples/dnx/twamp/cint_dnx_twamp.c`

NOTE: The example creates the TWAMP reflector over IPv4 application in `cint_dnx_twamp.c`, with the function `twamp_reflector_example()`.

To create the TWAMP-over-LAG reflector over an IPv4 application in the `cint_dnx_twamp.c` file, refer to the `twamp_reflector_lag_example()` function.

42.4 TWAMP Non-Accelerated

In this application, the device is configured to execute the role of the TWAMP TX and RX.

42.4.1 SOC Properties

None

42.4.2 Configuration Flow

- TWAMP TX non-accelerated
  1. Configure routing (My-Mac, RIF, VRF).
  2. Call `bcm_l3_ingress_create` to create FEC.
  3. Create IP route to FEC.
  4. Create the PMF by instructing the ETPP to timestamp transmitted TWAMP packets.
  5. The CPU constructs and injects packets with all the needed fields except for TX timestamp.
- TWAMP RX non-accelerated
  1. Create the PMF. Trap received TWAMP packets and redirect them to the Accelerator.
  2. CPU-RX. Decode and analysis in the CPU.

42.4.3 Shell Commands

None

42.4.4 Application Reference

CINT example with detailed description of the application.

- **Type:** CINT reference
- **Paths:**
  - `$SDK/src/examples/dnx/twamp/cint_dnx_twamp_field.c`
  - `$SDK/src/examples/dnx/twamp/cint_dnx_twamp.c`

NOTE: Creates the TWAMP TX/RX non-accelerated over IPv4 application in file `cint_dnx_twamp.c`, function `twamp_tx_rx_example()`

42.5 TWAMP Accelerated

In this application, the JR2 device is configured to execute the role of the TWAMP TX/RX.

42.5.1 SOC Properties

None

42.5.2 Configuration Flow

- TWAMP Accelerated TX
  - Configure routing (My-Mac, RIF, VRF).
  - Call `bcm_l3_ingress_create` to create FEC.
  - Create IP route → FEC.
  - Create the PMF.

    Instruct ETPP to timestamp transmitted TWAMP packets.
  - Configure the accelerated GTF.
  - The Accelerator constructs and injects packets with all the needed fields except for TX timestamp.
- TWAMP Accelerated RX
  - Configure the accelerated CTF.
  - Clean the accelerated CTF statistics.
  - Create the PMF:

    Trap received TWAMP packets and redirect them to the Accelerator.
  - Collect the accelerated CTF statistics.

42.5.3 Shell Commands

None

42.5.4 Application Reference

CINT example with detailed description of the application:

- **Type**: CINT reference
- **Path**:
  - `$SDK/src/examples/dnx/twamp/cint_dnx_twamp_field.c`
  - `$SDK/src/examples/dnx/twamp/cint_dnx_twamp.c`

NOTE: To create the TWAMP TX/RX accelerated over IPv4/IPv6 application in the file `cint_dnx_twamp.c`, refer to the function `twamp_tx_rx_example()`.

42.5.5 API Descriptions

42.5.5.1 bcm_switch_reflector_*

API Name	Highlights
`bcm_switch_reflector_create()`	Create `encap_id` for TWAMP reflector
`bcm_switch_reflector_get()`	Get TWAMP reflector data
`bcm_switch_reflector_destroy()`	Destroy TWAMP reflector
`bcm_switch_reflector_traverse()`	Traverse all TWAMP reflectors

Chapter 43: PON Application

43.1 Introduction

The PON application involves either network-facing NNI ports or MAC-facing PON ports. Packets on the NNI ports are Ethernet packets that are tagged with zero, one, or two VLAN tags. The VLAN tags identify the user and service (ONU) and class of service. Packets on the PON ports are Ethernet packets with an outermost VLAN tag that encodes the E-PON Logical Link Identification (LLID) or G-PON Encapsulation Mode Identification (GEMID). That tag is known as the *PON_TAG*. The PON_TAG is composed of PON_SubPort_ID and PON_Channel_ID. The *PON Mode* (that is, 4 × GPON, 2 × 10G EPON, and so on) is a PON interface attribute that defines the partition of PON_TAG bits [15:0] between PON_SubPort_ID and PON_Channel_ID.

The DNX switches support the following two mapping modes:

- PON_SubPort_ID is indicated on PON_TAG[15:12] and PON_Channel_ID is indicated on PON_TAG[11:0].
- PON_SubPort_ID is indicated on PON_TAG[14:13] and PON_Channel_ID is indicated on PON_TAG[12:0].

The PON_TAG is prepended to the VLAN-tag stack (as the outermost tag). The PON_TAG is always present on packets on PON ports, which may have a C-Tag and S-Tag as well. When a packet is forwarded downstream from an NNI port to PON ports, the incoming VLAN header from the NNI port may be retained, stripped, or modified, and a PON_TAG is inserted. When a packet is forwarded upstream from a PON port to NNI ports, the incoming VLAN header from the PON port may be retained, stripped, or modified, and the PON_TAG is stripped.

At the DNX ingress device, when a packet is forwarded upstream from the PON port to the NNI port, the PON_TAG is moved to the beginning of the packet.

The device classifies packets coming from PON ports into InLIFs and services them according to the mapped-pp-port, PON_TAG, incoming (up to two) VLAN tags, and (optionally) PCP and EtherType. All classification schemes can coincide on a single PON port and PON_TAG.

At the DNX egress device, when a packet is forwarded downstream from an NNI port to a PON port, the PON_TAG is encapsulated to the outermost VLAN tag.

Traffic on the OLT is in one of the following main categories:

- 1:1 Model
- N:1 Model

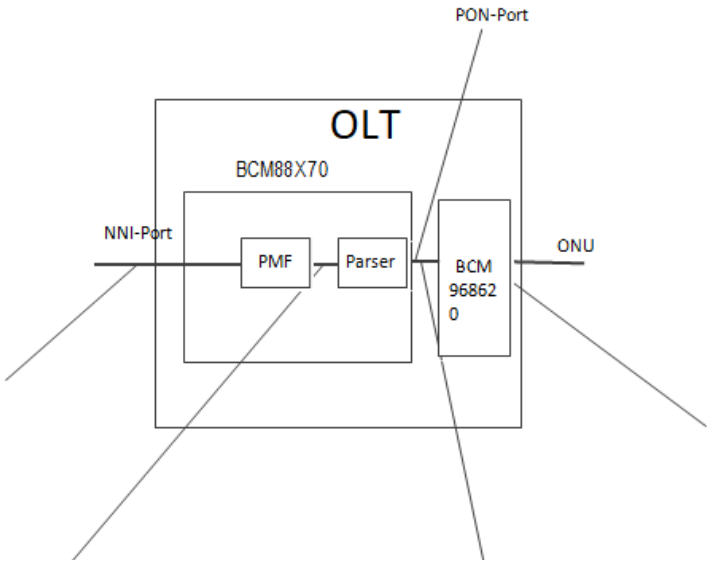
The 1:1 model is that of a P2P service. A downstream packet (NNI → PON) is classified as an NNI AC-LIF that points directly to the PON port and its PON_TAG encapsulation. The S-Tag is removed, the C-Tag is translated, and the PON_TAG is added. Likewise, an upstream packet (PON → NNI) is classified according to the PON port, PON_TAG, and VLAN tags to an AC-PON-LIF that points directly to the NNI port. PON_TAG is stripped, C-Tag is either kept or translated, and an S-Tag can be added.

In the N:1 model, a downstream packet (NNI → PON) is classified to a VSI. A lookup is performed based on the DA. If the DA is unknown, the packet is sent across a multicast group that contains the PON ports, or it is dropped.

Upstream packets are classified to VSI. An L2 SA is learned, and AC based learning is performed. If the DA is unknown, the packet is sent across a multicast group that contains the NNI port.

An *unmatched service* is a service model where a packet is classified to an unmatched service by the PON_TAG only (irrespective of VLAN tags). Unmatched service space is a distinct services space from the space that applies to *regular* untagged packets.

Figure 39: Packet Flow in OLT



The device supports the following lookups on ISEM-A/ISEM-B for LIF and service resolution per PON ports and PON_TAG:

- A (PON_UNMATCHED): {Mapped-pp-port(10), PON_TAG(16)}
- B1 (PON_UNTAGGED): {Mapped-pp-port(10), PON_TAG(16)}
- B2 (PON_SINGLE_TAG): {Mapped-pp-port(10), Tunnel-ID(16)}, VID1(12)}
- C (PON_DOUBLE_TAG): {Mapped-pp-port(10), Tunnel-ID(16)}, VID1(12), VID2(12)}

NOTE: VID1 stands for the Outer-VLAN-Tag-ID, and VID2 stands for the Inner-VLAN-Tag-ID. VID1 and VID2 can both be S-TAG or C-TAG.

PON_UNMATCHED uses the flag BCM_VLAN_PORT_MATCH_PORT_TUNNEL and PON_UNTAGGED uses the flag BCM_VLAN_PORT_MATCH_PORT_TUNNEL_UNTAGGED.

- Untagged packet arrives: Looks up the *untagged DB.* If there is no hit, it goes to the *unmatched DB.*
- Tagged packet arrives: Looks up one or more of the *tagged DB.* If there is no hit, it goes to the *unmatched DB.*

There are two PON classification modes known as STC (single tag classification) and DTC (double tag classification). If the PON port is configured as STC, it only supports (A, B) lookups on ISEM. If the PON port is configured as DTC, it supports (A, B) lookups on ISEM and C lookup on ISEM. If a packet matches multiple lookups, the more specific lookup takes priority. The following table summarizes the LIF matching options in the PON port according to the incoming packet tag structure.

Table 96: PON-LIF Matching Lookup Options per Incoming Tag-Structure

Incoming Packet Tag Structure	First Priority	Second Priority	Third Priority	Fourth Priority
Double Tags	{PON_TAG, VID1, VID2}	{PON_TAG, VID1}	PON_TAG (unmatched)	Port
Single Tag	—	{PON_TAG, VID1}	PON_TAG (unmatched)	Port
Untagged	—	PON_TAG (untagged)	PON_TAG (unmatched)	Port

43.2 Application Configuration Checklist

- PON port settings
- Create PON LIFs
- Create PON LIF classifications.
- Add PON LIF to VSI
- Create PON ingress/egress VLAN translation.
- Define multicast forwarding
- PON anti-spoofing settings

43.2.1 SOC Properties

None

43.2.2 N:1 Service Configuration Flow

To enable the PON port, call `bcm_port_control_set(unit, port, type, value)`

- `port`: PON port.
- `type`:

Supported Flags	Description
`bcmPortControlPONEnable`	If set, the PON application is enabled on this port.

- value:
  - 0: Disable
  - 1: Default mapping mode. PON_SubPort_ID is indicated on PON_TAG[15:12] and PON_Channel_ID is indicated on PON_TAG[11:0].
  - 2: Additional mapping mode. PON_SubPort_ID is indicated on PON_TAG[14:13] and PON_Channel_ID is indicated on PON_TAG[12:0].

To map `local_port` + `PON_SubPort_ID` to `mapped_pp_port`, call
`bcm_port_extender_mapping_info_set(unit, flags, type, *mapping_info)`

- `flags`: Both `BCM_PORT_EXTENDER_MAPPING_INGRESS` and `BCM_PORT_EXTENDER_MAPPING_EGRESS` should be set.
- `type`:

Supported Flags	Description
`bcmPortExtenderMappingTypePonTunnel`	If set, the PON application is enabled on this port.

- `mapping_info`: Holds the mapping information
  - `pp_port`: The `pp_port` being mapped to. Return from `bcm_port_get()`.
  - `tunnel_id`: PON_SubPort_ID from PON_TAG[15:12]
  - `phy_port`: PON port PTC
  - `vlan`: Unused

To configure PON-LIF lookup modes as STC/DTC, call `bcm_vlan_control_port_set(unit, port, type, arg)`

- `port`: PON port.
- `type`: `bcmVlanPortDoubleLookupEnable`

Supported Flags	Description
bcmVlanPortDoubleLookupEnable	If set, the PON application support the double-tag lookup.

- arg:
  - 0: PON STC port
  - 1: PON DTC port

To create VSI and flooding multicast groups, call the relevant API, as follows:

- To create a Multipoint Q-in-Q L2VPN, call bcm_vswitch_create(unit, *vsi)
  - vsi: The output allocated Vswitch VSI.
- To create multicast group for upstream, call bcm_multicast_create(unit, flags, *group)
  - flags: BCM_MULTICAST_INGRESS_GROUP and BCM_MULTICAST_WITH_ID should be set.
  - group: Allocated multicast group ID.
- To create a multicast group for downstream with additional offset, call bcm_multicast_create(unit, flags, *group)
  - flags: BCM_MULTICAST_INGRESS_GROUP and BCM_MULTICAST_WITH_ID should be set.
  - group: Allocated multicast group ID.

To configure PON-LIFs, call bcm_vlan_port_create(unit, *vlan_port)

NOTE: This information also applies to configuring an NNI-LIF.

The bcm_vlan_port_t structure configures both PON LIF and Network facing LIF. This document describes only the parameters relevant to PON application.

- Criteria:

Supported Criteria	Description
BCM_VLAN_PORT_MATCH_PORT_TUNNEL	Match by {Port, PON_TAG}
BCM_VLAN_PORT_MATCH_PORT_TUNNEL_UNTAGGED	Match by {Port, PON_TAG}
BCM_VLAN_PORT_MATCH_PORT_TUNNEL_VLAN	Match by {Port, PON_TAG, Outer-SVID}
BCM_VLAN_PORT_MATCH_PORT_TUNNEL_CVLAN	Match by {Port, PON_TAG, Outer-CVID}
BCM_VLAN_PORT_MATCH_PORT_TUNNEL_VLAN_STACKED	Match by {Port, PON_TAG, Outer-SVID, Inner-CVID}
BCM_VLAN_PORT_MATCH_PORT_TUNNEL_CVLAN_STACKED	Match by {Port, PON_TAG, Outer-CVID, Inner-CVID}
BCM_VLAN_PORT_MATCH_PORT_TUNNEL_SVLAN_STACKED	Match by {Port, PON_TAG, Outer-SVID, Inner-SVID}

- match_tunnel_value – Tunnel value to match
- egress_tunnel_value – Egress tunnel value
- match_vlan – Outer VLAN ID to match
- match_inner_vlan – Inner VLAN ID to match
- egress_vlan – Egress Outer VLAN
- egress_inner_vlan – Egress Inner VLAN

To associate a PON LIF with a VPN, call bcm_vswitch_port_add(unit, vsi, port).

NOTE: This information also applies to associating an NNI LIF with a VPN.

- vsi – The Vswitch VSI to which the AC-LIF is associated.
- port – The gport of associated AC-LIF.

To add PON LIF to multicast group, call `bcm_multicast_add(unit, group, flags, nof_replications, *rep_array)`

- `group` – MC group/bitmap ID provided upon group creation.
- `flags`:

Supported Criteria	Description
`BCM_MULTICAST_INGRESS_GROUP`	ingress multicast group/bitmap
`BCM_MULTICAST_EGRESS_GROUP`	ingress multicast group/bitmap

- `nof_replications`: `replication_array` size
- `rep_array`:
  - `flags`: Not used
  - `port`: Destination gport to be added for the n replication
    - System gport – Used for ingress MC
    - Flow-id gport – Used for ingress MC
    - Local gport – Used for egress MC
    - Trunk gport – Used for ingress and egress MC
    - group gport– Used to link a multicast group to other multicast group. Created by `BCM_GPORT_MCAST_GROUP_ID_SET` macro
    - bitmap gport – Used to link a multicast group to a multicast bitmap, as a sub-group. Created using `BCM_GPORT_MCAST_BITMAP_ID_SET` macro
  - `encap1`: Copy-unique-data (CUD) used for the n-th replication
  - `encap2`: Unused

To set PON LIF ingress VLAN editor, call one of the following APIs:

- To create action id, call `bcm_vlan_translate_action_id_set(unit, flags, action_id, *action)`
  - `flags`:
    - `BCM_VLAN_ACTION_SET_INGRESS` – The configuration is applicable for the Ingress side.
    - `BCM_VLAN_ACTION_SET_EGRESS` – The configuration is applicable for the Egress side.
    - `BCM_VLAN_ACTION_SET_WITH_ID` – The user specifies an unallocated VLAN Edit Command ID in the parameter `action_id`
  - `action_id` – The allocated VLAN Edit command ID to be set
  - `action` – The action to apply on the referenced
- To create new outer/inner VLAN and `vlan_edit_class_id` for PON LIF, call `bcm_vlan_port_translation_set(unit, *vlan_port_translation)`
  - `vlan_port_translation`:

    `flags`:
    - `BCM_VLAN_ACTION_SET_INGRESS` – The configuration is applicable for the Ingress side.
    - `BCM_VLAN_ACTION_SET_EGRESS` – The configuration is applicable for the Egress side
  - `gport` – The AC object for which the VLAN Edit attributes are configured
  - `new_outer_vlan` – A VID value for a constructed Outer Tag. Can be used also for an Inner tag. Any value in the 4K range is valid.
  - `new_inner_vlan` – A VID value for a constructed Inner Tag. Can be used also for an Outer tag. Any value in the 4K range is valid. Invalid for a Virtual Egress Default gport (zero value should be set).
  - `vlan_edit_class_id` – The VLAN Edit Profile to be used for VLAN Edit command mapping. For the number of VLAN Edit Profiles for Ingress and Egress

- To set an action ID for a specified combination of packet format ID and VLAN translation profile, call `bcm_vlan_translate_action_class_set(unit, *action_class)`
  - action_class:

    flags:
    - `BCM_VLAN_ACTION_SET_INGRESS` – The configuration is applicable for the Ingress side.
    - `BCM_VLAN_ACTION_SET_EGRESS` – The configuration is applicable for the Egress side
  - `vlan_edit_class_id` – The mapped VLAN-Edit Profile
  - `tag_format_class_id` – The mapped Tag format
  - `vlan_translation_action_id` – The mapping result VLAN Edit Command ID

NOTE: The preceding information also applies to setting a PON LIF egress VLAN editor.

Add an NNI LIF to a multicast group. Call the API as described previously.

Set the multicast group offset to flood downstream packets in multicast group, call `bcm_port_control_set(unit, port, type, value)`.

- port: PON port.
- type:

Supported Criteria	Description
`bcmPortControlFloodUnknownUcastGroup`	Flooding group for unknown unicast
`bcmPortControlFloodUnknownMcastGroup`	Flooding group for unknown multicast
`bcmPortControlFloodBroadcastGroup`	Flooding group for broadcast

- `value`: 0: disable, 1: enable.

43.2.3 1:1 Service Configuration Flow

1. Enable the PON port, call API as 1:1 service.
2. Mapping PTC + `PON_SubPort_ID` to `pon_pp_port`, call API as 1:1 service.
3. Configure PON-LIF lookup modes as STC/DTC, call API as 1:1 service.
4. Configure PON-LIFs, call API as N:1 service.
5. Set PON LIF ingress VLAN editor, call API as 1:1 service.
6. Set PON LIF egress VLAN editor, call API as 1:1 service.
7. Configure NNI-LIF, call API as 1:1 service.
8. Cross connect the PON-LIF and NNI-LIF, call `bcm_vswitch_cross_connect_add(unit, *gports)`
   - `gports`:
     - port1: PON-LIF
     - port2: NNI_LIF

For a calling sequence example, refer to `cint_pon_appliction.c`.

43.2.4 Anti-Spoofing

43.2.4.1 IP Anti-spoofing

IP anti-spoofing is a security feature for PON applications. The feature provides a mechanism to prevent user IP address spoofing by discarding upstream L3 packets received from ONUs that do not match the configured or DHCP-discovered source IP address. If the packet hits one host or subnet entry, it passes the pipeline. Otherwise, it can be dropped or trapped to the CPU for further analysis.

PON IP anti-spoofing supports the following two modes:

- DHCP mode: Lookup {SA, SIP, PON-LIF} – Configure by `bcm_l3_source_bind_add()` with SA is non-zero.
- Static mode: Lookup {SIP, PON-LIF} – Configure by `bcm_l3_source_bind_add()` with SA is zero.

43.2.4.2 MAC Anti-spoofing

MAC anti-spoofing is a security feature for PON applications. The feature provides a mechanism to prevent user MAC address spoofing by discarding upstream L2 packets received from ONUs that do not match the configured source MAC address. If the packet hits the MAC entry, it passes the pipeline. Otherwise, it can be dropped or trapped to the CPU for further analysis.

MAC Anti-Spoofing supports one mode. Lookup {SA} – Configure by `bcm_l2_addr_add()` with SA is non-zero and with the flag `BCM_L2_MOVE_PORT`.

43.2.5 Shell Commands

None

43.2.6 Application Reference

For an example of N:1 service Configuration Flow, refer to the following:

- **Type:** CINT reference
- **Path:** `$SDK/src/examples/dnx/pon/cint_dnx_pon_application.c`
- **Main function:** `pon_n_1_service()`.

For an example of 1:1 service Configuration Flow, refer to the following:

- **Type:** CINT reference
- **Path:** `$SDK/src/examples/dnx/pon/cint_dnx_pon_application.c`
- **Main function:** `pon_1_1_service()`.

For an example of IP anti-spoofing Configuration Flow, refer to the following:

- **Type:** CINT reference
- **Path:** `$SDK/src/examples/dnx/pon/cint_dnx_pon_ip_anti_spoofing.c`
- **Main function:**
  - `pon_service_ip_anti_spoofing_enable()`
  - `pon_service_ip_anti_spoofing_set()`
  - `cint_sav_field_config_ipmf1()`
  - `cint_sav_field_set_context_ipmf1()`

For an example of MAC anti-spoofing Configuration Flow, refer to the following:

- **Type:** CINT reference
- **Path:** `$SDK/src/examples/dnx/pon/cint_dnx_pon_mac_sav.c`
- **Main function:**
  - `bcm_port_class_set()`
  - `pon_mac_sav_add()`
  - `cint_field_pon_mac_sav_main()`

43.3 API Descriptions

43.3.1 bcm_port_control_*

API Name	Highlights
`bcm_port_control_set(unit, pon_port, bcmPortControlPONEnable, value)`	Enable the PON port. ■ 0: Disable PON port mapping. ■ 1: Default mapping mode. PON_SubPort_ID is indicated on PON_TAG[15:12] and PON_Channel_ID is indicated on PON_TAG[11:0]. ■ 2: Additional mapping mode. PON_SubPort_ID is indicated on PON_TAG[14:13] and PON_Channel_ID is indicated on PON_TAG[12:0].
`bcm_port_control_get(unit, pon_port, bcmPortControlPONEnable, &value)`	Get the PON port.

43.3.2 bcm_port_extender_mapping_info_*

API Name	Highlights
`bcm_port_extender_mapping_info_set(unit, flags, bcmPortExtenderMappingTypePonTunnel, &mapping_info)`	Mapping PTC + PON_SubPort_ID to PON_PP_PORT.
`bcm_port_extender_mapping_info_get(unit, flags, bcmPortExtenderMappingTypePonTunnel, &mapping_info)`	Get the PON_PP_PORT from PTC + PON_SubPort_ID.

43.3.3 bcm_vlan_control_port_*

API Name	Highlights
`bcm_vlan_control_port_set(unit, pon_port, bcmVlanPortDoubleLookupEnable, value)`	Configure the PON mode as STC/DTC.
`bcm_vlan_control_port_get(unit, pon_port, bcmVlanPortDoubleLookupEnable, &value)`	Get the PON mode.

43.3.4 bcm_vlan_port_*

API Name	Highlights
`bcm_vlan_port_create()`	Configure the PON LIF.
`bcm_vlan_port_destroy()`	Destroy the PON LIF.
`bcm_vlan_port_find()`	Find the PON LIF.

43.3.5 bcm_port_match_*

API Name	Highlights
`bcm_port_match_add()`	Configure the Multiple matches to the same PON LIF.
`bcm_port_match_delete()`	Remove a match from the PON LIF.

43.3.6 bcm_l3_source_bind_enable_*

API Name	Highlights
`bcm_l3_source_bind_enable_set(unit, pon_gport, enable)`	Enable the PON IP anti-spoofing per PON LIF.
`bcm_l3_source_bind_enable_get(unit, pon_gport, &enable)`	Get the PON IP anti-spoofing per PON LIF.

43.3.7 API bcm_l3_source_bind_*

API Name	Highlights
`bcm_l3_source_bind_add(unit, &info)`	Add an anti-spoofing entry.
`bcm_l3_source_bind_get(unit, &info)`	Get an anti-spoofing entry.
`bcm_l3_source_bind_delete(unit, &info)`	Delete an anti-spoofing entry
`bcm_l3_source_bind_delete_all(unit)`	Delete all anti-spoofing entries.
`bcm_l3_source_bind_traverse(unit, cb, &userdata)`	Traverse anti-spoofing entries.

Chapter 44: PPPoE

44.1 Introduction

Point-to-Point Protocol over Ethernet (PPPoE) combines the Point-to-Point Protocol (PPP), commonly used in dial-up connections, with the Ethernet protocol, which supports multiple users in a local area network. The PPP protocol information is encapsulated within an Ethernet frame.

PPPoE has two kinds of traffic: Discovery and Session. The Discovery stage traffic establishes connections between hosts and a service provider. When Discovery completes successfully, both the host and the service provider have the information necessary to build their PPP connection over Ethernet, which allows them to enter the PPP Session stage. Control traffic at the Discovery stage and data control traffic at the Session stage are both supposed to be trapped to the control plane for further processing, while data traffic at the Session stage should be handled by the data plane.

44.1.1 PPPoE Session Data

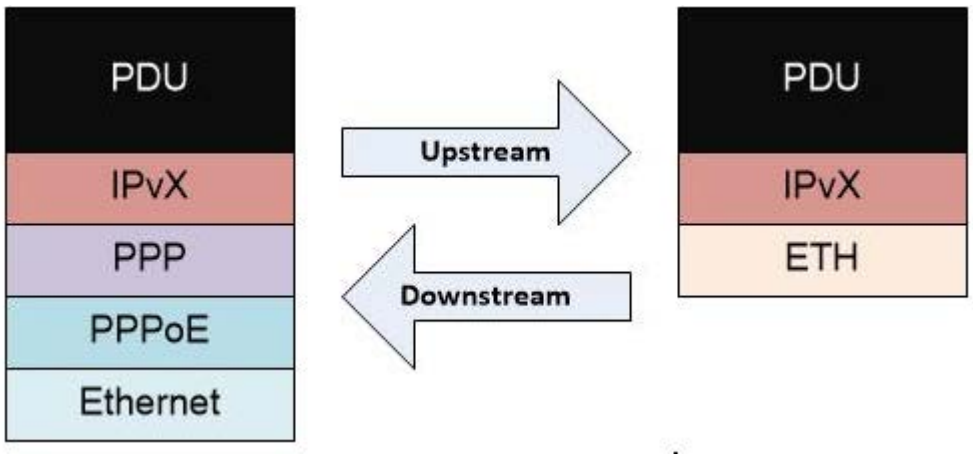
The PPPoE fields are as follows:

- VER(4b): *Must* be set to 0x1 for this version of the PPPoE specification
- TYPE(4b): *Must* be set to 0x1 for this version of the PPPoE specification
- CODE(8b): Discovery – various; PPP Session – 0x0
- SESSION_ID(16b): Unsigned value; value is fixed for a given PPP session
- LENGTH(16b): Length of the PPPoE payload. It does not include the Ethernet or PPPoE headers
- Protocol ID: Indicates protocol carried by the PPP, it can beIPv4 and IPv6

44.1.2 PPPoE MP

For termination, the PPPoE layer and PPP layer are terminated. The forwarding header is the inner protocol. For encapsulation, the PPPoE layer is built with the PPP layer.

Figure 40: PPPoE MP Example

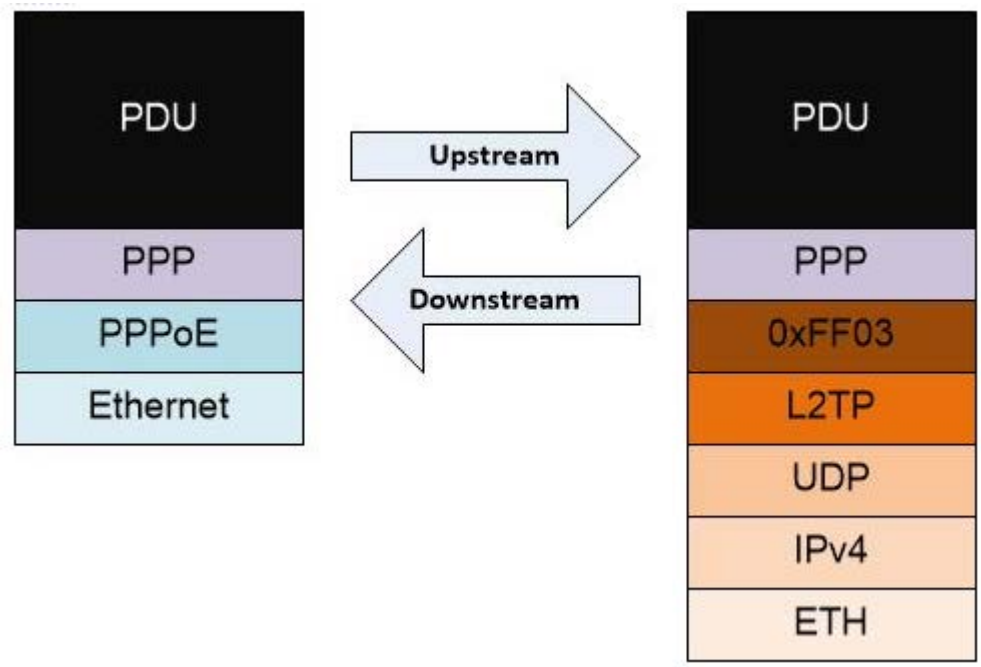


44.1.3 PPPoE P2P

For termination, only the PPPoE layer is terminated. The PPPoE Session LIF provides the destination. For encapsulation, the PPPoE layer is built without the PPP layer.

NOTE: PPPoE P2P mode is often used with L2TPv2 in a *cross-connect* scenario. More details are available in Chapter 45, L2TPv2 Encapsulation.

Figure 41: PPPoE P2P Example



44.2 Application Configuration Checklist

Bring-up Bridge-Router basic application including:

- Port properties
- My-MAC Layer 2 Termination
- ETH-RIF (L3 VSI interface) and VRF definition
- L3 Egress object: ARP with AC information
- Add forwarding entries, such as Host and Route.
- All information is available in Chapter 22, IP Router.

PPPoE MP termination:

- PPPoE MP termination with session spoofing check
- PPPoE MP termination with SIPv4 anti-spoofing
- PPPoE MP termination with statistics

PPPoE MP encapsulation:

- Create PPPoE encapsulation object, next OutLIF is ARP.
- Create L3 Egress object: FEC to point to PPPoE encapsulation objects. See Chapter 22, IP Router.
- Add Forwarding entries such as Host and Route. See Chapter 22, IP Router.
- PPPoE MP encapsulation with egress QOS

NOTE: PPPoE P2P mode is often used with L2TPv2 in *cross-connect* scenario, more details are in Chapter 45, L2TPv2 Encapsulation.

44.3 PPPoE MP Termination

The PPPoE session ID and source MAC address identify a PPPoE session.

Additional properties of PPPoE termination object, like VRF, can be set by calling `bcm_l3_ingress_create(int unit, bcm_l3_ingress_t *ing_intf, bcm_if_t *intf_id)`.

44.3.1 SOC Properties

`custom_feature_pppoe_session_spoofing_check_keep_dummy_lif`

PPPoE must allocate a pair of LIFs with the same higher 18 bits used in a PPPoE session spoofing check. A temporary LIF exists when the lower 2 bits of the first allocated LIF are not 0x00. The deallocation of this temporary LIF decreases the performance of the following LIF allocation.

If `custom_feature_pppoe_session_spoofing_check_keep_dummy_lif` = 1, it indicates to avoid deallocating the temporary LIF, if any. By default, the value is 0, and the temporary LIF is deallocated.

44.3.2 Configuration Flow

To create a PPPoE termination object, call `bcm_ppp_terminator_create(unit, bcm_ppp_terminator_t *info)`

- `info.flags` – See the following `BCM_PPP_TERM_XXX` flags

Supported Flags	Description
`BCM_PPP_TERM_WITH_ID`	If set, object id is provided by user
`BCM_PPP_TERM_REPLACE`	If set, update the encapsulation entry
`BCM_PPP_TERM_STAT_ENABLE`	If set, enable statistics
`BCM_PPP_TERM_WIDE`	Unsupported InLIF wide data feature is not available for PPP termination LIF
`BCM_PPP_TERM_SESSION_ANTI_SPOOFING`	If set, enable session spoofing check
`BCM_PPP_TERM_CROSS_CONNECT`	If set, enable P2P mode

- `info.type` – `bcmPPPTypePPPoE` for PPPoE
- `info.session_id` – PPPoE session ID
- `info.src_mac` – Source MAC address
- `info.ingress_qos_model` – Ingress QOS and TTL model, see more information in Chapter 10, QoS.
- `info.ppp_terminator_id` – PPPoE termination object ID

To enable a PPPoE session spoofing check, set the `BCM_PPP_TERM_SESSION_ANTI_SPOOFING` flag when calling `bcm_ppp_terminator_create(int unit, bcm_ppp_initiator_t *info)`.

In addition to creating PPPoE termination objects, add the PPPoE session spoofing check entries by using `bcm_ppp_term_spoofing_check_add(int unit, bcm_ppp_term_spoofing_check_t *info)`.

- `info.flags` – See `BCM_PPP_TERM_SPOOFING_CHECK_xxx`. Currently it is not defined.
- `info.type` – Spoofing check flags

Supported Flags	Description
`bcmPPPTermSpoofingCheckTypeVlanPort`	If set, spoofing checks if packets arrived from the expected incoming VLAN port.

- `info.ppp_terminator_id` – PPPoE terminator ID
- `info.vlan_port_id` – VLAN Gport ID

To enable SIPv4 anti-spoofing, configure a URFP to PPPoE termination object by calling `bcm_l3_ingress_create(unit, ing_intf, intf_id)`.If statistics are enabled, map stat ID and stat profile to the PPPoE termination object by calling the API `bcm_gport_stat_set(unit, gport, core_id, engine_source, stat_info)`, where `gport` is the PPPoE termination object ID. For more information about this API, see Chapter 15, PP Statistics Generation.

44.3.3 Shell Commands

None

44.3.4 Application Reference

Example of Basic-termination for PPPoE tunnel

- **Type:** CINT reference
- **Path:** `$SDK/src/examples/dnx/ppp/cint_dnx_pppoe.c`
- **Main function:** `pppoe_example_termination()`.

Example of termination for PPPoE tunnel with session spoofing check

- **Type:** CINT reference
- **Path:** `$SDK/src/examples/dnx/ppp/cint_dnx_pppoe.c`
- **Main function:** `pppoe_example_termination_session_spoofing_check()`.

Example of termination for PPPoE tunnel with SIPv4 spoofing check

- **Type:** CINT reference
- **Path:** `$SDK/src/examples/dnx/ppp/cint_dnx_pppoe.c`
- **Main function:** `pppoe_example_termination_sipv4_spoofing_check()`.

44.4 PPPoE MP Encapsulation

The PPPoE encapsulation object stores a session ID to build PPPoE header and the next OutLIF pointer for the following encapsulation. The PPPoE encapsulation object ID is used as a Tunnel-Pointer by the ingress forwarding databases (such as the FEC database). For information about how to create an FEC and point to the PPPoE encapsulation object (like any other Tunnel-pointer), see Section 22.8, L3 Egress Object: Ingress-FEC.

44.4.1 SOC Properties

None

44.4.2 Configuration Flow

To create a PPPoE encapsulation object, call `bcm_ppp_initiator_create(unit, bcm_ppp_initiator_t *info)`

- `info.flags` – See `BCM_PPP_INIT_XXX` flags

Supported Flags	Description
`BCM_PPP_INIT_WITH_ID`	If set, object id is provided by user
`BCM_PPP_INIT_REPLACE`	If set, update the encapsulation entry
`BCM_PPP_INIT_STAT_ENABLE`	If set, indicates statistics enabled

- `info.type` – `bcmPPPTypePPPoE` for PPPoE
- `info.session_id` – PPPoE session ID
- `info.encap_access` – Encapsulation phase to be used. The following phases are allowable for PPPoE. For more information about EEDB phase, see Section 7.4, EEDB Management.

Supported Encapsulation Phase	Description
`bcmEncapAccessRif`	Associated with logical phase 1
`bcmEncapAccessNativeArp`	Associated with logical phase 2
`bcmEncapAccessTunnel1`	Associated with logical phase 3

- `info.egress_qos_model` – Egress QOS and TTL model. For more information, see Chapter 10, QoS. To configure an egress QOS map, use the `bcm_qos_map_create(unit, flags, map_id)` and `bcm_qos_map_add(unit, flags, map, map_id)` APIs. Then, use the `bcm_qos_port_map_set(unit, port, ing_map, egr_map)` API to set a QOS profile where the port is a PPPoE encapsulation object.
- `info.l3_intf_id` – Next pointer interface ID to the link to an additional encapsulation entry, which is normally ARP.
- `info.ppp_initiator_id` – PPPoE encapsulation object ID.

If statistics are enabled, map the stat ID and stat profile to a PPPoE encapsulation object by calling the `bcm_gport_stat_set(unit, gport, core_id, engine_source, stat_info)` API, where `gport` is the PPPoE encapsulation object ID. For more information about the API, see Chapter 15, PP Statistics Generation.

44.4.3 Shell Commands

None

44.4.4 Application Reference

Example of Basic-encapsulation for PPPoE tunnel

- **Type:** CINT reference
- **Path:** `$SDK/src/examples/dnx/ppp/cint_dnx_pppoe.c`
- **Main function:** `pppoe_example_encapsulation()`.

Example of encapsulation for PPPoE tunnel with egress QOS

- **Type:** CINT reference
- **Path:** `$SDK/src/examples/dnx/ppp/cint_dnx_pppoe.`
- **Main function:** `pppoe_example_encapsulation_qos()`.

44.5 API Descriptions

44.5.1 bcm_ppp_initiator_*

API Name	Highlights
`bcm_ppp_initiator_create()`	Create PPPoE encapsulation object and set its properties.
`bcm_ppp_initiator_delete()`	Destroy PPPoE encapsulation object per given object ID.
`bcm_ppp_initiator_get()`	Retrieve PPPoE encapsulation object per given object ID.
`bcm_ppp_initiator_traverse()`	Traverse over all PPPoE encapsulation objects.

44.5.2 bcm_ppp_terminator_*

API Name	Highlights
`bcm_ppp_terminator_create()`	Create PPPoE termination object and set its properties.
`bcm_ppp_terminator_delete()`	Destroy PPPoE termination object per given object ID.
`bcm_ppp_terminator_get()`	Retrieve PPPoE termination object per given object ID.
`bcm_ppp_terminator_traverse()`	Traverse over all PPPoE termination objects.

44.5.3 bcm_dnx_ppp_term_spoofing_check_*

API Name	Highlights
`bcm_ppp_term_spoofing_check_add()`	Add a PPP Term spoofing check object per given PPP Tunnel Gport and VLAN Gport.
`bcm_ppp_term_spoofing_check_delete()`	Delete a PPP Term spoofing check object per given PPP Tunnel Gport and VLAN Gport.
`bcm_ppp_term_spoofing_check_traverse()`	Traverse over all PPP Term spoofing check objects.

Chapter 45: L2TPv2 Encapsulation

45.1 Introduction

The L2TPv2 encapsulation application defines how packets are switched into an L2TPv2 domain. This chapter provides information about how to configure L2TPv2 egress tunnel encapsulation and what actions occur after the forwarding on the main header (IP). This chapter also describes the API calls for defining L2TPv2 egress tunnels. For IPv4 egress tunnels, see Chapter 29, IP Tunnel v4 Encapsulation.

To gain the most value from this chapter, read the following sections first:

- The basic Packet Processor concepts of Switch/Router Abstraction. This section provides an overview of the basic L2 and L3 objects.
- Section 22.6, ETH-RIF (L3 VSI Interface) and VRF Definition.
- Chapter 22, IP Router This section provides information about configurations that are related to next hop information, namely FEC and Egress-ARP. Also, it provides information relevant for the MyMAC operation which is a prerequisite for IP tunnel forwarding and encapsulation.
- Chapter 29, IP Tunnel v4 Encapsulation. This section provides information about configurations that are related to IPv4 tunnel encapsulation.

45.2 Application Configuration Checklist

- Set L3 port properties – For more information, see Section 22.3, Port Routing Properties.
- Set ETH-RIF properties and create My-MAC per ETH-RIF – For more information, see Section 22.6, ETH-RIF (L3 VSI Interface) and VRF Definition.
- Create an egress object: Egress-ARP – For more information, see Section 22.7, L3 Egress Object: Egress-ARP (Next-Hop).
- Create an egress IP-tunnel encapsulation object – For more information, see Chapter 29, IP Tunnel v4 Encapsulation.
- Egress L2TPv2 Tunnel encapsulation object.

45.3 Egress L2TPv2 Tunnel Encapsulation Object

45.3.1 SOC Properties

None

45.3.2 Configuration Flow

To configure the egress L2TPv2 tunnel, call `bcm_ppp_initiator_create(unit, &tunnel)`

- `tunnel.type` – `bcmPPPTypeL2TPv2` for L2TPv2 tunnel encapsulation.
- `tunnel.flags`

Supported Flags	Description
`BCM_PPP_INIT_WITH_ID`	Create a L2TPv2 tunnel initiator with LIF ID passed by `tunnel.ppp_initiator_id`.
`BCM_PPP_INIT_REPLACE`	Replace an already allocated L2TPv2 tunnel initiator entry. The flag can be used only with the `BCM_PPP_INIT_WITH_ID` flag and a valid tunnel initiator ID.
`BCM_PPP_INIT_STAT_ENABLE`	Enable statistics counting for the L2TPv2 tunnel initiator LIF object. **NOTE:** It is not possible to move between STAT disable/enable upon replace.

- `tunnel.l2tpv2_tunnel_id` – Configure the L2TPv2 tunnel ID to be encapsulated.
- `tunnel.session_id` – Configure the L2TPv2 session ID to be encapsulated.
- `tunnel.l3_intf_id` – Specify the IPv4 tunnel initiator global LIF ID for UDPoIPv4oETH encapsulation.
- `tunnel.encap_access` – Specify from which encapsulation stage the tunnel encapsulation procedure will start.

NOTE: The L2TP encapsulation stage must be in an access phase lower than the IPv4 tunnel.

Supported Types	Description
`bcmEncapAccessRif`	Access local OutLIF at EEDB phase 1
`bcmEncapAccessNativeArp`	Access local OutLIF at EEDB phase 2
`bcmEncapAccessTunnel1`	Access local OutLIF at EEDB phase 3
`bcmEncapAccessTunnel2`	Access local OutLIF at EEDB phase 4
`bcmEncapAccessTunnel3`	Access local OutLIF at EEDB phase 5
`bcmEncapAccessTunnel4`	Access local OutLIF at EEDB phase 6
`bcmEncapAccessArp`	Access local OutLIF at EEDB phase 7

- `tunnel.egress_qos_model`: Configure tunnel QOS-Model and TTL-model. For more information see Chapter 10, QoS.

If statistics are enabled, configure their settings by calling the API `bcm_gport_stat_set(unit, gport, core_id, engine_source, stat_info)` where `gport` is the egress L2TPv2 tunnel LIF ID. For more information about the API, see Chapter 15, PP Statistics Generation.

To configure egress remark profile associated with the L2TPv2 tunnel LIF ID, call the APIs `bcm_qos_map_create(unit, flags, egr_map)`, where flags are configured – `BCM_QOS_MAP_EGRESS`, `BCM_QOS_MAP_REMARK`. To associate the L2TPv2 tunnel LIF ID with the remark profile call `bcm_qos_port_map_set(unit, port, ing_map, egr_map)`, where port is `tunnel.tunnel_id`. Ingress map is not required and can be left 0. For more detailed information, see Chapter 10, QoS.

45.3.3 Shell Commands

None

45.3.4 Application Reference

Example of Basic-encapsulation API usage for L2TPv2 tunnel

- **Type:** CINT reference
- **Path:** `$SDK/src/examples/dnx/ppp/cint_dnx_l2tp.c`

45.4 API Descriptions

45.4.1 Egress L2TPv2 Tunnel Encapsulation Object

API Name	Highlights
`bcm_ppp_initiator_create()`	Create Egress L2TPv2 tunnel and set its properties.
`bcm_ppp_initiator_delete()`	Destroy Egress L2TPv2 tunnel initiator given Tunnel LIF-ID.
`bcm_ppp_initiator_get()`	Retrieve Egress L2TPv2 tunnel properties given Tunnel LIF-ID.
`bcm_ppp_initiator_traverse()`	Traverse over all Egress L2TPv2 tunnel LIF objects.

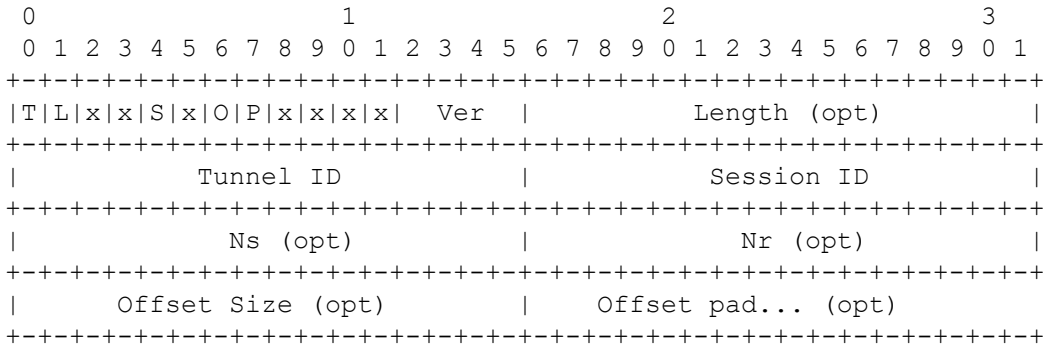
Chapter 46: L2TPv2 Termination

46.1 Introduction

The L2TPv2 termination application defines the means by which packets are switched out of an L2TPv2 domain.

The following figure shows the L2TPV2 header format.

Figure 42: L2TPV2 Header



The following bit options are supported:

- Length (L) bit: If L=1, the Length field is present.
- Sequence (S) bit: If S=1 the Ns and Nr fields are present.
- Offset (O) bit: If O=1, the Offset Size field is present.
- Priority (P) bit If P=1, this data message should receive preferential treatment in its local queuing and transmission.

L2TPv2 control message can be identified by a T bit with value of 1. This kind of traffic must be trapped and tunneled to the control plane as control traffic (to be implemented by the PMF).

NOTE: When the Offset (O) bit is 1, `offset_size` can be only 0x0. Other values are not supported, and offset pad octets do not exist.

This chapter provides information about how to configure L2TPv2 tunnel termination and what actions are taken prior to the forwarding on the main header (IP). The chapter refers mainly to how the tunnel termination stages are configured. It also describes the API calls for defining L2TPv2 ingress tunnels. For information about IPv4 tunnel termination, see Chapter 28, IP Tunnel v4 Termination.

Prior to reading this chapter, familiarize yourself with the following information:

- The basic Packet Processor concepts of Switch/Router Abstraction. This section provides an overview of the basic L2 and L3 objects.
- Section 22.6, ETH-RIF (L3 VSI Interface) and VRF Definition
- Chapter 22, IP Router. This section provides information about configurations that are related to next hop information, namely FEC and Egress-ARP. Also, it provides information relevant for the My-MAC operation which is a prerequisite for IP tunnel forwarding and termination.
- Chapter 28, IP Tunnel v4 Termination. This section provides information about configurations that are related to IPv4 tunnel termination.

46.1.1 Application Configuration Checklist

- Set L3 port properties – For more information, see Section 22.3, Port Routing Properties.
- Set ETH-RIF properties and create My-MAC per ETH-RIF – For more information, see Section 22.6, ETH-RIF (L3 VSI Interface) and VRF Definition.
- Ingress IP-tunnel termination object.
- Ingress L2TPv2 Tunnel termination object.
- Set Ingress L2TPv2 Tunnel session spoofing.

46.2 Ingress L2TPv2 Tunnel Termination Object

46.2.1 SOC Properties

None

46.2.2 Configuration Flow

To configure an ingress L2TPv2 tunnel, call `bcm_ppp_terminator_create(unit, &tunnel)`

- `tunnel.type` – `bcmPPPTypeL2TPv2` for L2TPv2 tunnel termination.
- `tunnel.flags`

Supported Flags	Description
`BCM_PPP_TERM_WITH_ID`	Create an L2TPv2 tunnel terminator with a Global LIF ID passed by the tunnel to `ppp_terminator_id`.
`BCM_PPP_TERM_REPLACE`	Replace an already allocated tunnel terminator entry. The flag can be used only with the `BCM_PPP_TERM_WITH_ID` flag and a valid `tunnel_id`.
`BCM_PPP_TERM_STAT_ENABLE`	Enable statistics counting for the tunnel termination LIF object.
`BCM_PPP_TERM_SESSION_ANTI_SPOOFING`	Create a tunnel terminator with session anti-spoofing enabled.
`BCM_PPP_TERM_CROSS_CONNECT`	Create a P2P tunnel terminator.

- `tunnel.network_domain` – Configure the network domain as part of the lookup key.
- `tunnel.l2tpv2_tunnel_id` – Configure the L2TPv2 tunnel id as part of the lookup key.
- `tunnel.session_id` – Configure the L2TPv2 session id as part of the lookup key.
- `tunnel.ingress_qos_model` – Configure the tunnel QOS-Model and TTL-model. For more information, see Chapter 10, QoS.

If statistics are enabled, configure their settings by calling `bcm_gport_stat_set(unit, gport, core_id, engine_source, stat_info)` where `gport` is the L2TPv2 Tunnel termination object ID. For more information about the API, see Chapter 15, PP Statistics Generation.

46.2.3 Shell Commands

None

46.3 Ingress L2TPv2 Tunnel Session Spoofing

Users can check the L2TPv2 tunnel ID and session ID of L2TPv2 packets that arrive from the *correct interface*. The L2TPv2 tunnel ID and session ID check is performed by doing a L2TPv2 terminator database lookup. The correct interface check is performed by doing an IP tunnel database lookup.

46.3.1 SOC Properties

None

46.3.2 Configuration Flow

To configure L2TPv2 tunnel session spoofing:

- Call `bcm_ppp_terminator_create(unit, &tunnel)`, where
  - `tunnel.flags` – `BCM_PPP_TERM_SESSION_ANTI_SPOOFING` for L2TPv2 Tunnel with session anti-spoofing enabled.
- Call `bcm_ppp_term_spoofing_check_add(unit, &term_spoofing)`, where
  - `term_spoofing.vlan_port_id` – Configure the IP tunnel LIF as part of the lookup key.
  - `term_spoofing.ppp_terminator_id` – Configure the L2TP terminator LIF as part of the lookup key.

46.3.3 Shell Commands

None

46.3.4 Application Reference

Example of Basic-termination and session spoofing APIs usage for L2TPv2 tunnel

- **Type:** CINT reference
- **Path:** `$SDK/src/examples/dnx/ppp/cint_dnx_l2tp.c`

46.4 API Descriptions

46.4.1 Egress L2TPv2 Tunnel Termination Object

API Name	Highlights
`bcm_ppp_terminator_create()`	Create Ingress L2TPv2 tunnel and set its properties.
`bcm_ppp_terminator_delete()`	Destroy Ingress L2TPv2 tunnel given tunnel LIF-ID.
`bcm_ppp_terminator_get()`	Retrieve Ingress L2TPv2 tunnel properties given tunnel LIF-ID.
`bcm_ppp_terminator_traverse()`	Traverse over all Ingress L2TPv2 tunnel LIF objects.

Chapter 47: GPRS Tunneling Protocol

47.1 Introduction

GPRS Tunneling Protocol (GTP) is a group of IP-based communications protocols for carrying general packet radio service (GPRS). Commonly, GTP is used in the User Plane Function (UPF) units, which are also related to the 3GPP 5G architecture to facilitate user-plane operation. GTP is a tunnel protocol divided into GTP-U, GTP-C, and GTP. Each GPD-U tunnel is identified by its tunnel endpoint ID (TEID).

This application is under development. For the current status, contact your Broadcom representative.

Chapter 48: Instrumentation IPT (INT and Tail-Edit)

48.1 Introduction

The Instrumentation IPT profile is required for metadata on instrumentation applications (Alternate Marking). The IPT profile controls the metadata that is collected and built in the egress.

The inserted data is composed of device (node) metadata

The IPT profile is also used by the IFA applications in the field processor.

48.2 IPT Profile Properties

48.2.1 SOC Properties

None

48.2.2 Configuration Flow

Set IPT profile configurations:

```
bcm_instru_ipt_profile_set(int unit, uint32 flags, int ipt_profile,bcm_instru_ipt_t * config)
```

- `flags` – Irrelevant. Set to 0.
- `ipt_profile` – Profile ID. Valid only for profile 1.
- `config.node_type`.

For metadata header information:

- `config.metadata_flags`.

48.2.3 Application Reference

Example of PMF configurations for INT application

- **Type:** CINT reference
- **Path:** `$SDK/src/examples/dnx/mpls/cint_mpls_deep_stack_alternate_marking.c`

48.3 Node ID (Switch ID)

One of the metadata fields is node ID (switch ID). This section describes how to set.

48.3.1 SOC Properties

None

48.3.2 Configuration Flow

Set switch ID (later to be used in the metadata header):

```
bcm_instru_control_set(unit, 0, bcmInstruControlIptSwitchId, switch_id);
```

48.4 Trace Probability

To perform statistical operations in PMF, one of the qualifiers should be the `trace_probability` bit. This bit is statistically generated (set to 1) according to a preconfigured probability per source port as below.

48.4.1 SOC Properties

None

48.4.2 Configuration Flow

Set trace probability:

```
bcm_instru_gport_control_set(unit, gport, 0, bcmInstruGportControlTraceProbability, percentage);
```

- `gport` – Source port gport (local port)
- `percentage` – Percentage in units of one-tenth. Meaning that, for example, a probability of 40% is achieved by setting this value to 400.

48.5 ETPP Redirect to Recycle for IPT Packets

IPT packets (the packets with an INT application specific FTMH extension) can be trapped in ETPP and recycled back for the second path. The mechanism which is used to recycle the packet is ETPP trap recycling (for more info go to traps UM section).

48.5.1 Configuration Flow

Enable/disable trap to recycle for egress port:

```
bcm_instru_gport_control_set(0, gport, 0, bcmInstruGportControlIptTrapToRcyEnable, arg);
```

- `gport` – Destination port gport, traffic destined to this port will be trapped and redirected to recycle interface.
- `arg` – 1 enable, 0 disable

48.6 IPT ACL Configurations

The IPT profile and queue information (INT application specific extension) are PMF actions.

- IPT profile – The IPT profile is PMF1 or PMF3 action (**bcmFieldActionIPTProfile**)
- Queue information – For ‘queue information application extension’ to be added to FTMH, iPMF3 should provide
- **bcmFieldActionIPTCommand** action (should be set to 1) .

48.6.1 SOC Properties

None

48.6.2 Configuration Flow

See Chapter 11, Field Processor for APIs.

48.6.3 Application Reference

Example of PMF configurations for INT application

- **Type:** CINT reference
- **Path**: `$SDK/src/examples/dnx/field/cint_field_hash_ipt.c.`

48.7 API Descriptions

48.7.1 IPT

API Name	Highlights
`bcm_instru_ipt_profile_set()`	Set IPT profiles attributes (metadata, first node header, and so on).
`bcm_instru_ipt_profile_get()`	Get IPT profiles attributes (metadata, first node header, and so on)
`bcm_instru_control_set()`	Set IPT switch ID (Node ID).
`bcm_instru_control_get()`	Get IPT switch ID (Node ID).

48.7.2 General

API Name	Highlights
`bcm_instru_gport_control_set()`	Set trace probability (can be used as PMF qualifier for statistical sampling). Set redirect for IPT packets.
`bcm_instru_gport_control_get()`	Get trace probability (can be used as PMF qualifier for statistical sampling). Get redirect for IPT packets.

Chapter 49: Instrumentation – Alternate Marking

49.1 RFC 8321 Alternate Marking Ingress Node

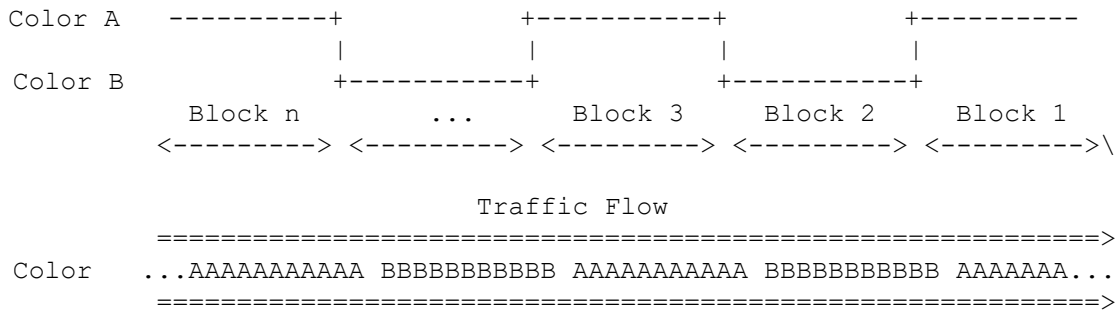
49.1.1 Introduction

RFC 8321 describes a method to measure packet loss, delay, and jitter on live traffic. This is achieved by Alternate-Marking (coloring) technique.

49.1.1.1 Loss Measurement

Traffic flows are virtually split into consecutive blocks where blocks are created by *coloring* traffic so that packets belonging on the same block have the same color. Whenever the color changes, the previous block terminates and a new one begins. Packets in each color are assigned a different counter, for example counter A and counter B, for each respective color. When counter A is active, counter B will be static and may be accurately sampled. In the following figure, during Block 3 it is possible to read counter B and get an accurate traffic count from Block 2. The color can be switched according to a fixed timer, at which stage counter A is static, and the traffic count from Block 4 can be gathered.

Figure 43: Computation of Link Packet Loss



The ingress node changes the color, and the egress (and possibly intermediate node) assign counter A or B according to the color of the incoming traffic.

In this implementation, the ingress node inserts in-band (for live traffic) a special MPLS label with the color information (the L bit), such that L=1 corresponds to color A (and counter A), and L=0 corresponds to color B.

During period B, the management can read counter A in ingress and egress nodes (which are stable), and compare the results to detect packet loss.

49.1.1.2 Delay Measurement

To measure the end-to-end latency between ingress and egress, the ingress node can add another (in-band for live traffic) information, called D.

If D=1 the ingress and egress nodes generate a copy of the packet, and sent it to the collector with Timestamp. For information about the format, see Chapter 48, Instrumentation IPT (INT and Tail-Edit).

By comparing the timestamps from ingress to egress, the management can determine the end-to-end latency.

To avoid flooding the collector with copies of the packets, D=1 is set by the ingress node only for the first packet immediately after alternating from one counter to another. This is done in the following way:

1. Ingress PMF assigns counter to AM packets.
2. Ingress PMF adds an OAM-ASE header with a dedicated AM OAM-subtype to AM packets and stamps the counters (assigned in the previous step) on the packet.
3. Egress PMF recognizes the OAM-ASE header with the AM subtype and counter 0 and assigns an alternate marking ACE context value. Counter value will be 0 for the first packet only (after toggling the **L**-bit). ePMF will also snoop these packets.
4. Micro code is selected based on the ACE in Step 3, which turns on the **D**-bit.

49.1.1.3 Identification and Encapsulation

Monitoring is done by intercepting traffic in the ingress node, adding a set of special labels, and then having egress and possibly intermediate nodes identify flows according to the special labels.

Both *flow-based strategies* and *link-based strategies* as described in RFC 8321 may be configured through field processor APIs.

The ingress node encapsulates in-band traffic with a *Special Label Identification* and a *Flow ID* label. The ingress node identifies packets to be monitored and snoops them and adds special labels. Alternate Marking (AM) traffic runs in parallel to data traffic. Egress and intermediate nodes use these labels to identify AM packets and assign counters to them.

Packets with the D bit enabled (the first packet in each flow) will be snooped, along with timestamp information.

The Special Label Identification indicates

the existence of the Flow-ID label. The label itself may be determined by the user.

The Flow ID is defined per flow (service) by the user.

The **L-bit**, and **D-bit** are described in Section 49.1.1.2, Delay Measurement.

R-bit is reserved and must be 0.

The BOS indication is the **B-bit**. It is set to 0 on the Special Label Identification and 1 on the Flow ID.

Two sets of flow labels should be created: one with the L bit and one without.

49.1.1.4 Alternate Marking

Alternating the **L-bit** per flow is done through a 32-bit global register defined in the iPMF1 called the *KeyGenVar* register. Each bit represents an **L-bit** profile for which coloring is done, and each flow will use one of these profiles.

The PMF qualifier is composed of a 5-tuple (or any other way of intercepting traffic) along with the KeyGenVar. Two entries are added into the TCAM per flow. Both entries are identical except for the L-bit-profile. One entry uses value 1 (entry A) and, the other entry uses value 0 (entry B). Thet of the KeyGenVar is masked.

Entry A’s action will assign OutLIF A and counter A, Entry B will assign OutLIF B and counter B.

OutLIF A and B will represent the two sets of flow labels and Special Label Identification which differ only in the L-bit.

Changing the color in the above flow is achieved by changing the respective bit in the KeyGenVar. This changes the color for all flows using the same L-bit-profile.

As mentioned, after alternating between one counter and another, it will be possible to read the inactive counter. One of the challenges is managing eviction:

- On the active counter, eviction must not be enabled. Otherwise **D** bit will be set on every packet, not just the first.
- On the passive counter, eviction must be set. Otherwise, it will not be possible to get the counter value

This can be resolved with one of two approaches:

1. Use different counters engines for **L** bit set/cleared. For example the counter A will be on engine X1 and counter B on engine X2. When switching from counter A to B by toggling the L bit:
   a. Enable eviction on counter engine X1 with `bcm_stat_counter_database_enable_set()`.
   b. Wait for eviction to complete. This may be be controlled through `bcmStatCounterSequentialSamplingInterval` in the API `bcm_stat_counter_engine_control_set()`
   c. Read counter A through
   d. `bcm_stat_counter_get()`. Use the flag `BCM_STAT_COUNTER_CLEAR_ON_READ`
   e. Disable eviction back on counter engine X1.
2. Use the same counter engine but different ranges for counter A, B and use `eviction_boundaries` to enable eviction on the passive counter only. When switching from counter A to B by toggling the L bit:
   a. Use `bcm_stat_database_eviction_boundaries_set()` to activate eviction only on the passive counter
   b. Enable eviction on the counter engine.
   c. Wait for eviction to complete as in the previous step.
   d. Get the counter as in the previous step.
   e. Disable eviction back on the counter engine.

For alternate marking, define two (alternate) time periods per flow: Period A and B. Usually, the duration of A and B are approximately the same. However, for different flows, it is possible to define different period lengths. For example, one flow period can be 10 seconds, while another flow period is 2 minutes. Multiple user-defined periods are supported.

49.1.2 Configuration Flow

The ingress node must configure five sets of APIs: encapsulation, counter processor, field processor, and snoop.

49.1.2.1 Encapsulation

The user is responsible for two labels:

- Special Label – Special Label Identification is configured through `bcm_switch_control_set()`. Set the type to `bcmSwitchMplsAlternateMarkingSpecialLabel` and the `arg` to the required label.
- Flow ID – Flow ID should be be configured twice: with and without the L-bit set. This is done with two calls to `bcm_mpls_tunnel_initiator_create()`
  – `label_array[0].label` should be the 20b Flow ID
  – `label_array[0].flags` should be set to `BCM_MPLS_EGRESS_LABEL_ALTERNATE_MARKING`. One call should be with the flag `BCM_MPLS_EGRESS_LABEL_ALTERNATE_MARKING_LOSS_SET` and one without
  – Other fields should be set according to needs, so documentation in relevant chapter.

49.1.2.2 Counter Processor

Setting the counter engine as for regular OAM counter with the source being `bcmStatCounterInterfaceIngressOam`. The only difference from standard OAM counters is that `bcm_stat_counter_eviction_set` should be called. `eviction_algorithmic_disable` should be set to 1.

49.1.2.3 Field Processor

Each Ingress AM node should have two iPMF1 entries: one entry represents the entry with **L** bit set, and one entry represents the entry with the **L** bit cleared. Field processor qualifier is based on 5-tuple and the KeyGenVar described above. Ingress PMF-1 should have the following actions:

1. LM statistics interface – For OAM stat-ID 7, 8, and 9 are available (`bcmFieldActionStatId7`, `bcmFieldActionStatId8`, `bcmFieldActionStatId9`).
2. Corresponding stat-profile (`bcmFieldActionStatProfile7`, `bcmFieldActionStatProfile8`, `bcmFieldActionStatProfile9`).
3. Add OutLIF with Flow ID. This is done with `bcmFieldActionOutVport0Raw`.
4. Add OAM-ASE header with AM-subtype. This is done through the action bcmFieldActionOamRaw
5. Set the LM-read index with the action `bcmFieldActionStatOamLM`. This is done used for stamping the counter on the ASE header. The value used depends on the stat-ID from the first action: stat-ID 7 maps to LM-read-index 0, Stat-ID 8 to 1, Stat-ID 9 to 2.

For each action, the following values should be used:

1. Value associated with `StatIdX` is the counter value itself. For eviction purposes it is recommended to have both counters for the same sessions as far away from each other in the engine as possible. Alternatively the counters should be on different engines.
2. Stat profile is composed of the following four parts:
   - `is_meter` (counter increment indication - must be 1)
   - `is_lm` (must be 1 since we are using OAM-LM counters)
   - `type_id` (must be 0)
   - `valid`, (must be 1).

   This action must be the same for both iPMF1 entries.
3. OutLIF is produced from `label_array[0].tunnel_id` from `bcm_mpls_tunnel_initiator_create()` in the encapsulation stage. Value should be set using the macro `BCM_L3_ITF_VAL_GET`. Each iPMF1 entry should use a different OutLIF according to the **L bit**.
4. OAM raw value should be 0xA0000 which means using the dedicated OAM subtype 10 which is then stamped on the snooped copy's ASE header. Other fields should be 0. This should be the same for both entries.
5. LM read index should be `bcmFieldStatOamLmIndex0`, `bcmFieldStatOamLmIndex1`, or `bcmFieldStatOamLmIndex2`, depending on which `StatId` action is used. Stat ID 7 uses LM Index 0; 8 is mapped to 1; 9 is mapped to 2. This should be the same for both entries.

The egress PMF is responsible for snooping the first packet after the **L bit** was toggled as well as setting a special ACE value for context selection that sets the **D bit** in the encapsulation.

1. Create the ACE format
   a. Initialize a `bcm_field_ace_format_info_t` struct.
   b. Set `nof_of_actions` to 1
   c. Set `action_types[0]` to `bcmFieldActionAceContextValue`
   d. Call `bcm_field_ace_format_add()` and get a `bcm_field_ace_format_t` number.
2. Add the ACE entry
   a. Initialize a `bcm_field_ace_entry_info_t` struct.
   b. Set `nof_entry_actions` to 1
   c. Set `entry_action[0].type` to `bcmFieldActionAceContextValue`
   d. Set `entry_action[0].value` to `bcmFieldAceContextAlternateMarking`
   e. Call `bcm_field_ace_entry_add()` with the flags being 0, the `bcm_field_ace_format_t` from the previous step, the `bcm_field_ace_entry_info_t` filled above and the output being a `entry_handle` number.
3. Create an egress PMF entry.
   a. The qualifier Should be based on `bcmFieldQualifyOamTsSystemHeader`. value[0] should be 0, value[1] should be 0xa000. mask[0] should be 0xFFFFFe00, mask[1] should be 0xFFFF. This represents selection according to ASE with subtype 0xA (special subtype for AM) and data 0. The data represents the packet counter since the **L bit** was altered. Assuming the counters were properly cleared the data will only be 0 for the first packet.
   b. ePMF should have three actions:
      - `bcmFieldActionAceEntryId`. The value should be the `entry_handle` from Step e.
      - `fieldActionSnoopRaw`
      - `fieldActionSnoopStrengthRaw`

49.1.2.4 Snoop

The ingress snoops packets it identifies and sets special values on the system headers. Snooped packets should come with the time-of-day for Alternate Marking flows.

The two ways of obtaining the ToD are as follows:

1. Through the IPT APIs that append a tail edit to the end of the packet
2. Through *mirror additional information*, which add 64 bytes from the original packet and time of day

When using the second method, the format of snooped packets will be {FTMH+system-headers, 52B reserved, 6B ingress-ToD, 6B egress-TOD, network-headers}. The reserved 52B includes original system headers. Existence of this stack is indicated by the FHEI of the first set of system headers (usually the trap code is EgTxFieldSnoop1). In the CINT, this is controlled through `tod_append_mode`. This process involves calling three sets of APIs:

- `bcm_mirror_destination_create()`
  - Set the `gport` to the required destination
  - Use the macro `BCM_GPORT_MIRROR_SET()` to set the `mirror_dest_id` with the snoop command used in the second bullet in Step b in the egress PMF configuration.
  - Set the flag `BCM_MIRROR_DEST_IS_SNOOP`
  - Set `packet_control_updates.valid` to `BCM_MIRROR_PKT_HEADER_UPDATE_FABRIC_HEADER_EDITING`
  - When using *mirror additional information*, add the flag `BCM_MIRROR_DEST_EGRESS_ADD_ORIG_SYSTEM_HEADER`

- `bcm_mirror_header_info_set()`
  - Set `pp.tail_edit_profile` to 1
  - Set `tm.ase_ext.ase_type` to `bcmPktDnxAseTypeInt`
  - Set `tm.ase_ext.valid` to 1
  - When using *mirror additional information*, call the API with all the parameters zero.

49.1.2.5 IPT

An Instru-IPT profile ensures that ToD is stamped. Use the `bcm_instru_ipt_profile_set()` and `bcm_instru_control_set()` APIs discussed in Chapter 48, Instrumentation IPT (INT and Tail-Edit).

Ingress ToD timestamp is available only in the BCM88800, BCM88830, and BCM88480 devices. Egress ToD is available on all devices.

49.1.2.6 Alternate Marking

Alternating between **L bit** set and cleared is done by toggling the keyGenVar bit according to the session's profile. For an example, see `cint_field_alternate_marking_period_change()` in `cint_field_alternate_marking.c`.

The first packet after the **L bit** was switched will be snooped to the CPU with delay measurement information.

49.1.3 Shell Commands

None

49.1.4 Application Reference

A detailed cint appears in the files `cint_mpls_deep_stack_alternate_marking.c` and `cint_field_alternate_marking.c`.

- The main function is `mpls_deep_stack_alternate_marking_encapsulation_example()`. This calls all the functions mentioned in the configuration flow.
- Switching the **L bit** is done through `cint_field_alternate_marking_period_change()`
- The inactive counter can be read and cleared through `read_counter_and_handle_eviction()`

49.2 RFC 8321 Alternate Marking Intermediate and Egress Nodes

The roles of the intermediate and egress nodes are to assign a counter per Flow ID × L Bit and snoop packets with the D bit set.

Documentation can be found in the CINT files.

49.2.1 Configuration Flow

Similarly to the ingress node, the configuration has three parts: snoop, counter processor, and field processor.

49.2.1.1 Snoop

First, ingress snoop must be set up.

1. Set up the snoop command with `bcm_mirror_destination_create()`. The output will be the `mirror_dest_id` from the `bcm_mirror_destination_t` struct.
2. Create a user-defined trap with `bcm_rx_trap_type_create()` and `bcm_rx_trap_set()`. In the latter API, set `snoop_cmnd` in the `bcm_rx_trap_config_t` struct to be output from the first stage (with the `BCM_GPORT_MIRROR_GET()` macro). The output here is the `trap_id` from the former API.
3. Set the profile. This is done with `bcm_mirror_header_info_set()`.
   a. Use the flag `BCM_MIRROR_DEST_IS_SNOOP`
   b. `mirror_dest_is` should be the `trap_id` from the previous step.
   c. In the `mirror_header_info_t` struct, in `pp.tail_edit_profile`, set `tm.flow_id_ext.valid` to 1.
4. Instru-IPT profile.Use the `bcm_instru_ipt_profile_set()` and `bcm_instru_control_set()` APIs as mentioned in the IPT section.

49.2.1.2 Counter Processor

Counter processor may be set up as in the ingress node, however eviction should be enabled. This is because only the SW needs to read the counter, not the HW.

49.3 Application Reference

For an example of Alternate Marking with RFC8321, refer to the CINT the following files:

- `cint_mpls_deep_stack_alternate_marking.c`
- `cint_field_alternate_marking.c`.

The main function is `mpls_alternate_marking_termination()`. This calls all the functions mentioned in the configuration flow.

The inactive counter can be read and cleared through `mpls_deep_stack_alternate_marking_get_counter()`.

For an example for alternate marking with control word in intermediate node, a detailed CINT appears in the files `cint_vpls_alternate_marking.c`:

- The main function is `vpls_alternate_marking_hvpls_with_cw_example(...)`. This calls all the required functions mentioned in the configuration flow.
- Global variables `term_has_cw` and `encap_has_cw` indicate incoming packets with or without control word separately.

49.4 Alternate Marking over SRv6 (IPv6)

Alternate Marking on IPv6 in the ingress node is also supported. There are a few ways to support Alternate Marking over IPv6. Currently, Alternate Marking over IPv6 is supported for SRv6 only.

Alternate Marking can be based on the Flow Label of the IPv6 header, as follows:

- `Flow_label[15:0]` is the Flow-ID
- `Flow_label[16]` is the D bit
- `Flow_label[17]` is the L bit

49.4.1 Configuration Flow

The AM solution for a single-pass flow is composed of multiple steps. These steps are based on the MPLS solution.

1. This step occurs in iPMF2.

   Qualify the 5-tuple, similar to with MPLS. The action is the container (the content is `flow_id`).

2. This step occurs in iPMF3

   Qualify the container (`flow_id`) and `KeyGenVar`. The action is `TrunkHashKey`, `NetworkLoadBalanceKey`, `statID`, `OamRaw` (including `oam_subtype`), and `StatOamLM`.

   `Flow_lable[15:2]` are set into the network LB key, together with the L bit and D bit using `bcmFieldActionNetworkLoadBalanceKey`. `Flow_label[1:0]` are set into the trunk LB key using `bcmFieldActionTrunkHashKeySet`

3. This step occurs in ePMF.

   Qualify `oam_subtype` RFC8321Ipv6OnFlowId and `oam_counter_value` 0. The action generates a snoop copy. This is done for the first packet of every flow (L=1 or 0).

   The D-bit is updated automatically under the same condition.

   For two-pass (extended encapsulation solution) SRv6, this step is performed in the second pass, the rest are done in the first pass.

49.4.2 Application Reference

Two examples can be found in `cint_srv6_alternate_marking.c`.

- `srv6_alternate_marking_ingress_node_example` is for SRv6 packet with a one-pass solution.
- `srv6_ext_encap_alternate_marking_ingress_node_example` is for SRv6 packet with an extended encapsulation solution.

49.5 IFIT Ingress Node

In-situ Flow Information Telemetry (iFIT) performs functions similar to Alternate Marking but has flexible extendability. It utilizes existing and emerging on-path telemetry techniques to enable the collection and correlation of performance information from the living network.

For more information on Loss measurement, delay measurement, see Section 49.1, RFC 8321 Alternate Marking Ingress Node.

The alternate marking signaling in the iFIT header is similar to RFC 8321.

The Flow Instruction Indicator Label (FII) is equivalent to the Special Label Identification in RFC 8321. The S bit in FII is the MPLS BOS indication. The R bit is a reserved bit and should always be set to 1. The usage of the R/S bit in the flow ID header (FIH) depends on the S bit in FII. If the S bit is 0, the R/S bit is a BOS indication. Otherwise, the R/S bit is a reserved bit and should always be set to 1.

In this document, the iFIT is described based on the assumption that the FII label is a BOS label.

The alternate marking procedure and methodology in iFIT is identical to that of RFC 8321.

49.5.1 Configuration Flow

Similar to RFC 8321, the configuration is composed of encapsulation, counter processor, field processor, snoop, and IPT. The only difference between iFIT and RFC 8321 Alternate Marking is the encapsulation stage which is described below. For other parts, see Section 49.1.2, Configuration Flow.

49.5.1.1 Encapsulation

The user is responsible for two labels:

- Special Label – The flow instruction indication label is configured through `bcm_switch_control_set()` with type of `bcmSwitchMplsFlowInstructionIndicatorSpecialLabel` and the arg to be the required label.
- Flow ID – The flow ID should be configured twice, with and without the L-bit set. This is done with two calls to `bcm_instru_ifit_encap_create(int unit, bcm_instru_ifit_encap_info_t *ifit_encap_info);`
  - `ifit_encap_info` → `fih_flow_id` – The 20b Flow ID.
  - `ifit_encap_info` → `fih_r_s_bits` – Must be 1.
  - `ifit_encap_info` → `fii_exp` – EXP for the FII label. The value should be 0.
  - `ifit_encap_info` → `fii_ttl` – TTL for the FII label. The value should be 255.
  - `ifit_encap_info` → `fieh_length` – Indicate whether FIEH exists, and if so, indicate its size.
  - `ifit_encap_info` → `fih_header_type_indicator` – The header type in FIH.
  - `ifit_encap_info` → `fieh_ext_data` – FIEH data.
  - `ifit_encap_info` → `flags` – Flags, including the following:
    - `BCM_INSTRU_IFIT_ENCAP_WITH_ID`
    - `BCM_INSTRU_IFIT_ENCAP_REPLACE`
    - `BCM_INSTRU_IFIT_ENCAP_ALTERNATE_MARKING_LOSS_SET`

49.5.2 Application Reference

Example for Alternate Marking with iFIT.

- **Type:** CINT reference
- **Path:** `$SDK/src/examples/dnx/mpls/cint_mpls_deep_stack_ifit_am.c`
- **Function:** `mpls_deep_stack_ifit_rfc8321_encapsulation_example()`

49.6 IFIT Intermediate Node

The processing of iFIT in intermediate node for alternate marking is the same as with RFC 8321. The only difference is in the construction of User Defined Headers (UDH) if the intermediate node is a Super Provider Edge (SPE) node in which the entire iFIT header is kept after the packet is forwarded with the inner header.

In RFC 8321, the UDH is built with the special label and flow label totaling 64 bits in the MSB by ingress PMF. In iFIT, the UDH should be built with the flow label and its 96 bits following data. This is because the size of the iFIT header is variable depending on the contents.

49.6.1 Configuration Flow

The flow is similar to RFC 8321. For details, see Section 49.2.1, Configuration Flow.

49.6.2 Application Reference

Example of configurations for the iFIT SPE node.

- **Type:** CINT reference
- **Path:** `$SDK/src/examples/dnx/mpls/cint_mpls_deep_stack_alternate_marking.c`
- **Function:** `mpls_alternate_marking_termination()`

49.7 IFIT Egress Node

In the egress node, both the MPLS headers and iFIT header are terminated. Loss measurement and delay measurement occur in the same way as in the immediate node. For more details about the loss measurement and delay measurement, see Section 49.6, IFIT Intermediate Node.

However, due to the specific processing on termination of the iFIT header, two MPLS labels (at most) before the iFIT header can be terminated in one pipeline pass.

49.7.1 Configuration Flow

For loss measurement and delay measurement, see Section 49.6, IFIT Intermediate Node. For MPLS termination, including MPLS L2VPN and L3VPN, see Chapter 23, MPLS Label Switch Router.

There are no special configurations for iFIT header termination from the user's side. The default configuration handles it properly.

49.7.2 Application Reference

Example of configurations for iFIT in VPLS.

- **Type:** CINT reference
- **Path:** `$SDK/src/examples/dnx/utilities/cint_dnx_utils_vpls.c`
- **Function:** `vpls_main()`

Example of configurations for iFIT in MPLS L3VPN.

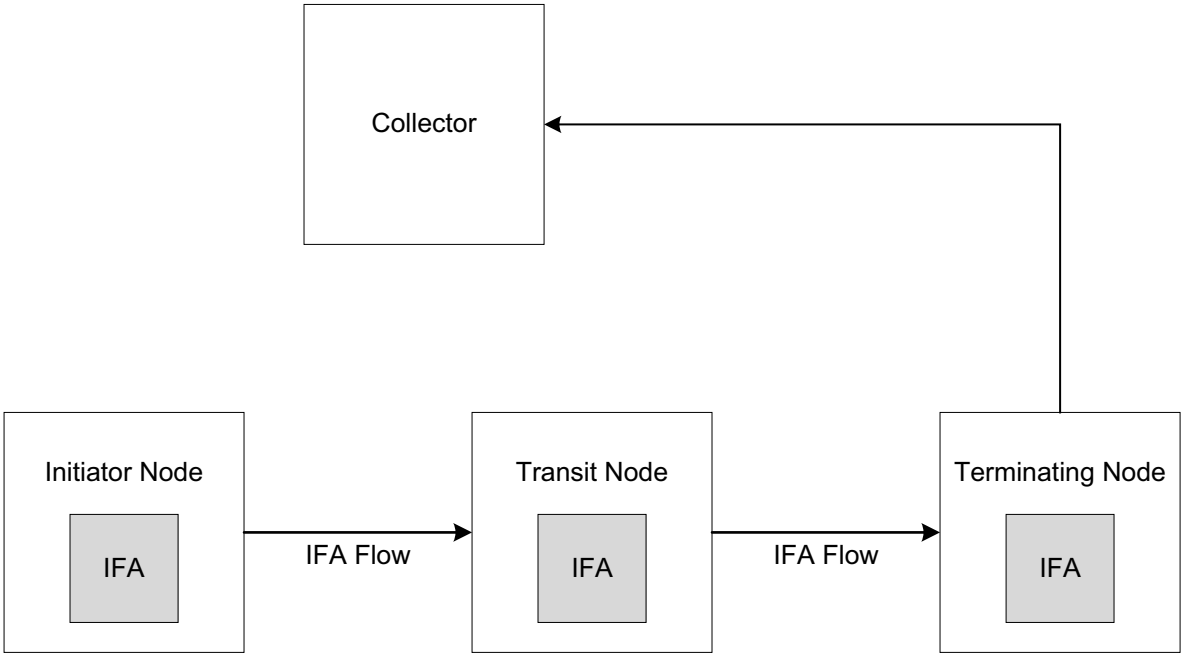
- **Type:** CINT reference
- **Path:** `$SDK/src/examples/sand/cint_mpls_termination_basic.c`
- **Function:** `mpls_termination_basic_main()`

Chapter 50: Instrumentation – Inband Flow Analyzer

50.1 Introduction

The Inband Flow Analyzer (IFA) records flow-specific information from a smart NIC or switches (or both) across the network. IFA defines an IFA header to mark the flow and mandate the collection of analyzed metadata on a per-marked packet, per-hop basis across the network.

Figure 44: IFA Flow



50.2 IFA 1.0

IFA 1.0 duplicates the sampled traffic and inserts metadata at all nodes in the analyzed network scope.

50.2.1 Initiator Function Node

The initiator function node is responsible for the following functions:

- Samples the flow traffic of interest based on a configuration.
- Converts the traffic into an IFA flow by adding an IFA header:
  - Updates the packet with initiator node metadata (Probe header).
  - Reinjects the IFA flow in the network.
  - Mandates (if necessary) a specific template ID metadata by all networking.
- Inserts the transit node metadata.

50.2.2 Transit Function Node

This node is responsible for inserting transit node metadata in the IFA packet.

50.2.3 Terminating Function Node

The terminating function node is responsible for the following functions:

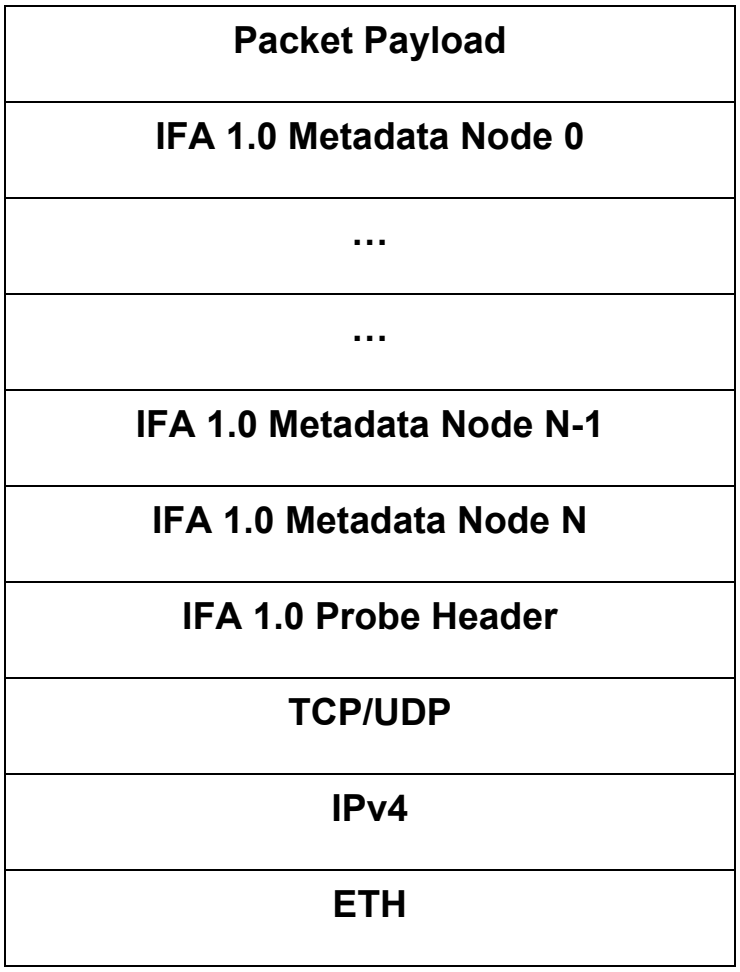
- Inserts terminating node metadata in the IFA packet (identical to transit node metadata).
- Terminates the IFA flow by summarizing the metadata of the entire path and send it to collector
- Drops the IFA flow.

50.2.4 IFA 1.0 Packet Encapsulation

In addition to the original packet's encapsulation, IFA flow packet encapsulation has an IFA Probe header and IFA node metadata for each node in the path of the packet, where node 0 is the initiator node, and node N is the last node.

Supported original packet encapsulations are ETH0/1/2 over IPv4 over TCP/UDP.

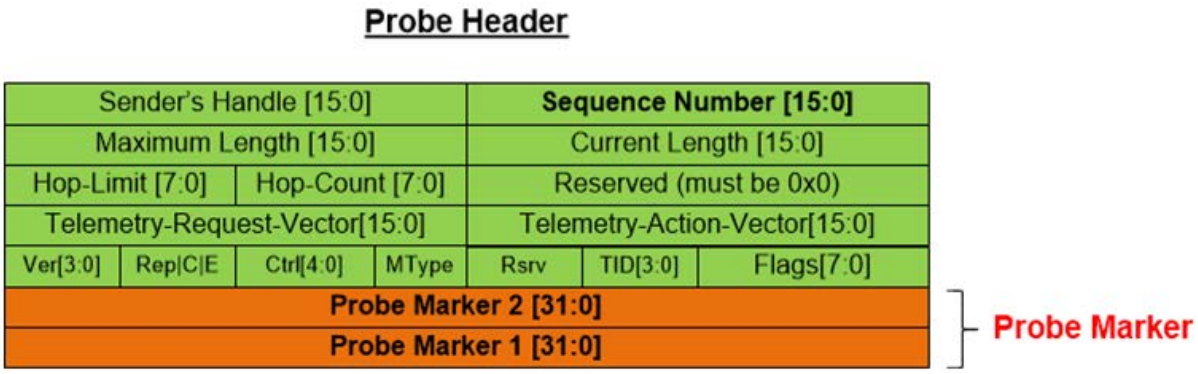
Figure 45: IFA Packet Encapsulation



50.2.5 IFA 1.0 Probe Header

The following figure shows the IFA probe header.

Figure 46: IFA Probe Header



The Probe header fields are as follows:

- Probe Marker – Arbitrary 32-bit values generally used by the network elements to identify the packet as a probe packet. These fields should be interpreted as unsigned integer values, stored in network byte order. The Probe marker's value is set through the `bcm_ifa_config_info_set` API. For transit and termination nodes, a value for this field should also be set in PMF1 configuration.
- Version Number – Currently set to 1.
- Rep – Replication requested. These bits indicate the replication of a packet. This field is used to explore all the valid forwarding paths and is set to 0.
- C– Copy requested. This bit is set for all the replicated packets to distinguish them from the original packets. This bit is set to 0.
- E – Maximum hop count exceeded. This bit is set when the device cannot add metadata because it has exceeded the hop count limit. Set to 0.
- Ctrl – These are the bits for local optimizations. Set to 0.
- Message Type – The value of this field can be either 1 (Probe) or 2 (Probe Reply). Set to 1.
- RSVD – Reserved bits, which are initialized to 0.
- TID (template ID) – The mandated template ID, which must be honored by all the networking elements in the path. The value of this field is set through the `bcm_ifa_config_info_set` API.
- Flags – This field is 8 bits. Set to 0.
- Telemetry Request Vector – A 16-bit field that requests well-known inband telemetry information from the network elements on the path. Set to 0x3b.
- Telemetry Action Vector – A 16-bit field that requests inband telemetry metadata to be inserted based on the action indicated from the network elements on the path. Set to 0.
- Hop Limit – This field is treated as an integer value representing the number of network elements. Its value is set through the `bcm_ifa_config_info_set` API.
- Hop Count – Specifies the current number of hops of capable network elements the packet has transit through. It begins with zero and will be incremented by one for every network element that adds a telemetry record.
- Max Length – Specifies the maximum length of the telemetry payload in bytes. Its value is set through the `bcm_ifa_config_info_set` API.
- Current Length – Specifies the current length of data stored in the probe. Updated in each node.
- Sender's Handle – Set by the sender to allow the receiver to identify a particular originator of probe packets. Along with the Sequence Number field, it enables the tracking of packet order and loss within the network. Its value is set through the `bcm_ifa_config_info_set` API.

50.2.6 References

Refer to the IETF IFA 1.0 specification: https://tools.ietf.org/html/draft-kumar-ifa-00.

50.2.7 Application Configuration Checklist

The application configuration checklist includes the following items:

- Global configuration
- Initiator node configuration
- Transit node configuration
- Terminator node configuration

50.2.8 Global Configuration

A global configuration consist of the configuration of flow-specific parameters, which is done through `bcm_ifa_config_info_set(unit, options, config_data)`.

Parameters to be configured are given as fields of `config_data`:

- `config_data.probemarker_1` – Used by the network elements to identify the packet as a probe packet.
- `config_data.probemarker_2` – Used by the network elements to identify the packet as a probe packet.
- `config_data.template_id` – TID – Mandated template ID.
- `config_data.device_id` – Used to identify the device reporting telemetry information.
- `config_data.hop_limit` – Represents the maximum number of network elements allowed.
- `config_data.max_payload_length` – Represents the maximum payload length of the IFA packet.
- `config_data.senders handle` – Set by the sender to allow the receiver to identify a particular originator of probe packets.

50.2.8.1 SOC Properties

None

50.2.8.2 Application Reference

IFA 1.0 parameter configuration:

- **Type**: CINT reference
- **Path**: `src/examples/dnx/instru/cint_ifa.c` – `cint_ifa1_node_parameters_config`

Counter resource allocation example:

- **Type**: CINT reference
- **Path**:
  - `src/examples/dnx/instru/cint_ifa.c` – `cint_set_IFA_counter`
  - `src/examples/dnx/oam/cint_oam_basic.c` – `set_counter_resource`

50.2.9 Initiator Node Configuration

The initiator node filters the traffic to create the IFA flow and adds the IFA probe header and IFA metadata of the initiator node.

50.2.9.1 SOC Properties

None

50.2.9.2 Configuration Flow

A detailed and accurate example may be found in `cint_instru_ifa1_initiator()`.

1. Global configuration:
   – Set node parameters including IFA session's probe-header using the `bcm_ifa_config_info_set` API.
   Example: `cint_ifa1_node_parameters_config`.
   – Configure a counter to be used for sequence number stamping, as in `cint_set_IFA_counter`. The counter should be allocated on the same core as the `recycle_port`.
   – Create IFA entity for the IFA header by using `bcm_instru_ifa_encap_create`.
   The IFA entity creates the OutLIF used as a pointer to the sequence number counter and couples the OutLIF the counter. This counter value is stamped on the IFA header.
   API Parameters:
   – `ifa_info.flags` – The flags in the following table are supported.

Flags	Description
`BCM_INSTRU_IFA_ENCAP_WITH_ID`	Specify the OutLIF-ID (`encap_id`) upon creation
`BCM_INSTRU_ENCAP_REPLACE`	Use in case of replace (must go with `WITH_ID` flag)

   – `ifa_info.stat_cmd` – Counter ID associated with the IFA session for sequence number
   – `ifa_info.counter_command_id` – Counter interface associated with the IFA session
   – `ifa_info.ifa_encap_id` – The returned OutLIF-ID for IFA header or input in case `WITH_ID` flag is set
2. Set up snooping, as in `cint_field_ip_snoop_set`. The snoop packet will point to the IFA header.
3. Set up field configurations, as in `cint_field_group_const_example_IFA_gen`
   Configure iPMF1:
   Set field group with two qualifiers and three actions.
   Qualifiers:
   a. Filter condition: Snoop filtering may be performed according to:
      - Flow (5-tuple or signature, or other definitions of *flow*)
      - Flow state (like queue level, drops, and so on)
      - Flow BW (elephant or mice)
      - Statistical sampling
      - Any combination of the preceding rules
   b. IFA Probe header

Actions:

a. Set a container for iPMF3

b. Snoop packet

c. Set tail edit profile = 3

– Set two TCAM entries:
  - On the filter condition, use action (b) for a snoop action. Attach to FirstIFA contexts in egress (To be used in the first cycle to encapsulate IFA probe header)
  - On the IFA Probe header condition, use actions (a) to signal iPMF3 to set INT and (c) to set Tail Edit Profile = 3. Attach to IFA contexts in egress (to be used in the second cycle to encapsulate IFA node metadata)

– Configure iPMF3 (used only for metadata attachment):
  - Qualifier: Container is set.
  - Action: Set INT = 1 (because INT cannot be set in iPMF1, it is done in iPMF3)

– Configure ePMF:
  - Qualifier: Destination port is recycle
  - Action: Set tail edit profile = 3 (This is done for correct egress context selection)

As a result in the first pass, the packet will hit iPMF1 and ePMF, and in the second pass it will hit iPMF1 and iPMF3.

50.2.9.3 Application Reference

Example of configuration snoop:

- **Type:** CINT example
- **Path:** `src/examples/dnx/field/cint_field_ip_snoop.c`
- **Name**: `cint_field_ip_snoop_set`

Example of initiator field configurations:

- **Type:** CINT example
- **Path:** `src/examples/dnx/instru/cint_field_IFA_datapath.c`
- **Name**: `cint_field_group_const_example_IFA_gen`

IFA entity creation: `bcm_instru_ifa_encap_create(unit,&ifa_info)`

IFA entity delete: `bcm_instru_ifa_encap_delete(unit,&ifa_info)`

IFA entity get: `bcm_instru_ifa_encap_get(unit,&ifa_info)`

IFA entity traverse: `bcm_instru_ifa_encap_traverse(unit, callback_function,&user_data)`

50.2.10 Transit Node Configuration

This node adds transit node metadata in the IFA packet.

50.2.10.1 Configuration Flow

1. Global configuration:
   a. Set Node parameters, including the IFA session's probe-header.
      Example: `cint_ifa1_node_parameters_config`.
2. Set up field configurations:
   a. Configure iPMF1:
      Set field group with one qualifier and two actions:
      Qualifiers:
      - IFA Probe Header

      Actions:
      - Set Container for iPMF3
      - Set Tail Edit Profile = 3

   b. Set TCAM entry:
      On the IFA Probe header qualifier, use the two actions and attach to IFA contexts in egress
   c. Configure iPMF3:
      - Qualifier: Container is set.
      - Action: Set INT = 1
        (Because INT cannot be set in iPMF1, it is done in iPMF3.)

50.2.10.2 Application Reference

- **Type:** CINT example
  Path: `src/examples/dnx/instru/cint_instru_IFA_datapath.c`
- **Name**: `cint_field_group_const_example_IFA_gen`

50.2.11 Termination Node Configuration

This node adds the terminator node's metadata in the IFA packet, sends the IFA packet to the collector, and terminates the IFA flow.

50.2.11.1 Configuration Flow

The first cycle is exactly the same as the transit node packet walk, so apply the transit node configuration. Additionally, set a terminator node configuration.

1. Configure the transit node.
2. Set up field configurations for the terminator node. Example:
   `cint_field_IFA_datapath_egress_main`
   a. Set the field group with on qualifier and one action:
      Qualifier: IFA Probe Header
      Action: Select IFA Trap Contexts
   b. Set the field group with two qualifiers and one action:
      Qualifiers:
      - IFA Probe Header
      - In port is recycle

      Action:
      - Trap packet

   – 2a. Set up ePMF (for first cycle)
   – 2b. Set up iPMF1 (for second cycle)
3. Enable an ETPP trap for IFA packets on the original destination port, sending them to recycle.
   Call `bcm_instru_gport_control_set(int unit, bcm_gport_t gport,uint32 flags, bcm_instru_gport_control_t type, int arg)`
   - `unit` – Device unit
   - `gport` – Original destination port
   - `flags` – Set to 0, not in use
   - `type` – Set to `bcmInstruGportControlIptTrapToRcyEnable`
   - `arg` – Egress port profile, set to 1

50.2.11.2 Application Reference

Refer to the following application:

- **Type:** CINT example
- **Path:** `src/examples/dnx/instru/cint_instru_IFA_egress_datapath.c`
- **Name**: `cint_field_IFA_datapath_egress_main`

50.2.12 API Descriptions

50.2.12.1 bcm_instru_ifa_encap_*

API Name	Highlights
`bcm_instru_ifa_encap_create()`	Create an IFA entity by coupling a designated OutLIF to a counter.
`bcm_instru_ifa_encap_delete()`	Delete an IFA entity
`bcm_instru_ifa_encap_get()`	Get counter information of an IFA entity.
`bcm_instru_ifa_encap_traverse()`	API for traversing IFA entities.

50.2.12.2 bcm_ifa_config_info_*

API Name	Highlights
`bcm_ifa_config_info_set()`	Configuration of IFA parameters of the device.
`bcm_ifa_config_info_get()`	Getting the IFA parameters of the device.

50.3 IFA 2.0

IFA 2.0 encapsulates analyzed metadata in the sampled packets at all nodes in its scope. On StrataDNX devices, IFA 2.0 is applied on live traffic.

IFA 2.0 is supported over IPv4 TCP/UDP only.

50.3.1 Initiator Function Node

The initiator function node does the following:

- Samples the flow traffic of interest based on a configuration.
- Converts the traffic into an IFA flow:
  - Encapsulates the IFA 2.0 header
  - Encapsulates and initializes the IFA 2.0 metadata header.
- Encapsulates the packet with initiator node metadata.
- Re-injects the IFA flow into the network.

50.3.2 Transit Function Node

This node is responsible for encapsulating transit node metadata in the IFA packet and updating metadata header.

50.3.3 Terminating Function Node

The terminating function node does the following:

- Terminates with metadata: Encapsulates the terminating node metadata in the IFA packet (identical to transit node metadata).
- Terminates without metadata: Does not encapsulate terminating node metadata in the IFA packet.

  A copy of the packet is sent to a collector for analysis.
- Terminates the IFA flow by stripping IFA headers and the metadata stack. The packet is then forwarded to its original destination.

50.3.4 IFA 2.0 Packet Encapsulation

The IFA 2.0 header is encapsulated as Layer 3, below the IP header, and its `next_layer` field points to Layer 4. Layer 4 is the TCP/UDP layer. The IFA 2.0 metadata header is positioned after TCP/UDP followed by the metadata stack. Each node encapsulates its metadata, on top of the metadata stack.

The fields in the IFA 2.0 header are as follows:

- Version (4 bits) – Specifies the version of IFA header. Value = 0x2.
- GNS (4 bits) – Global Name Space. Specifies the IFA zone scoped name space for IFA metadata. Value = 0xf.
- Protocol Type (8 bits) – IP header protocol type. This is copied from the IP header. Value = According to IP.Protocol

  IP.Protocol in IP layer is configurable. The default value = 253.

  NOTE: IP.Protocol value 253 is used for experimentation and testing. There is a chance that the value is also used for other applications. In this case, use BCM API `bcm_ifa_header_create` to configure IP.Protocol value for IFA 2.0.
- Flags (8 bits)
  - 0: R – Reserved. Value = 0x0
  - 1: R – Reserved. Value = 0x0
  - 2: R – Reserved. Value = 0x0
  - 3: MF – Metadata Fragment. Indicates the presence of the optional metadata fragment header. Value = 0x0
  - 4: TS – Tail Stamp. Indicates the IFA zone is requiring tail stamping of metadata. Value = 0x0
  - 5: I – Inband. Indicates this is live traffic. Strip and forward will be performed by the terminator node if this bit is set. Value = 0x1
  - 6: TA – Turn Around. Indicates that the IFA packet needs to be turned around at the terminating node of the IFA zone and sent back to source IP address. Value = 0x0
  - 7: C – Checksum. Indicates the presence of the optional checksum header. Value = 0x0
- Max Length (8 bits) – Specifies the maximum allowed length of the metadata stack in multiples of 4 octets. This field is initialized by the initiator node. Each node in the path will compare the current length with the maximum length, and if the current length equals or exceeds the maximum length, the transit nodes will stop inserting metadata.

  Value = configurable by `bcm_ifa_config_info_set (config_data.max_payload_length).`

The IFA 2.0 metadata fields are as follows:

- Request Vector (8 bits) – This vector specifies the presence of fields as specified by GNS. Fields are always 4-octet aligned. This field can be made extensible by defining a new GNS for an IFA zone. Set to 0xff.
- Action Vector (8 bits) – This vector specifies node-local or end-to-end action on the IFA packets.Each node in the path MAY use the action bitmap to insert or not insert the metadata based on exceeding a locally-specified threshold. Not inserting the metadata is indicated by setting the field value to –1 (all 1s). Set to 0x0.
- Hop Limit (8 bits) – Specifies the maximum allowed hops in an IFA zone. This field is initialized by the initiator node. The hop limit is decremented at each hop. If the incoming hop limit is 0, current nodes will not insert metadata. A value of 0xFF means that the hop limit check will be ignored.

  Value is configurable in the initiator node only by `bcm_ifa_config_info_set(config_data.hop_limit).`
- Current Length (8 bits) – Specifies the current length of the metadata in multiples of 4 octets.

  Set to 8 by initiator node, and incremented each node by 8 unless the maximum length or hop limit is reached.

The first 4 bytes of the metadata are the local namespace (LNS) + device ID. The rest of the metadata is for the given local namespace and global namespace.

IFA 1 and IFA 2 use the same metadata.

50.3.5 Initiator Node Configuration

The initiator node filters the traffic to create the IFA 2.0 flow. It also adds the IFA 2.0 header, IFA 2.0 metadata header, and the IFA 2.0 metadata of the initiator node.

Operation Principle

The traffic is sampled according to a 5-tuple. The sampled packet is set to tail-edit-profile=7, and the INT is selected for adding the IFA header, IFA metadata header, and metadata. The IP-protocol in IPv4 is set to IFA protocol-id. The next-header in the IFA 2.0 header will be copied from the original IP layer.

50.3.5.1 Configuration

A detailed and accurate example may be found in `cint_instru_ifa2_initiator()`.

1. Global configuration:
   – Set node parameters using the `bcm_ifa_config_info_set` API.
   Example: `cint_ifa2_node_parameters_config`.
   – Set IFA 2.0 header parameters using `bcm_ifa_header_create`.
   Example: `cint_config_IFA2_header`.
2. Set up field configurations, as in `cint_field_IFA2_datapath_initiator_main`

   Configure iPMF1:
   – Qualifiers:

     Filter condition according to:
     - Flow (5-tuple or signature, or other definitions of flow)
     - Flow state (like queue level, drops, and so on)
     - Flow BW (Elephant or mice)
     - Statistical sampling
     - Any combination of the above rules
   – Actions:
     - Set Container for iPMF3
     - Set Tail Edit Profile = 5
   – Set TCAM Entry:

     On filter condition:
     - Action to signal iPMF3 to set INT
     - Action to set Tail Edit Profile = 5

   Configure iPMF3
   - Qualifier: Container is set.
   - Action: Set INT = 1 (Because INT cannot be set in iPMF1 it is done in iPMF3)

Load Balancing Configuration

When IFA is used in the system, hashing of the *next-protocol* field in the IP header must be disabled by the parser. Instead, the user can add *next-header* for load-balancing keys using the ingress PMF hash mechanism for all flows.

Disable the parser ability of next-protocol through `bcm_control_set()` with the enum `bcmSwitchHashIP4OuterField`. Unset the bit `BCM_HASH_FIELD_PROTOCOL`. (Refer to the configuration in `cint_ifa2.c`).

iPMF1 should update the hashing keys for all traffic flows (not just IFA). For a detailed example, refer to `cint_ifa2_hash_configuration()` in `cint_ifa2.c`.

This configuration applies only to intermediate and termination nodes.

50.3.5.2 Application Reference

Example of initiator configuration steps:

- **Type:** CINT example
- **Path:** `src/examples/dnx/instru/cint_ifa2.c`
- **Name**: `cint_instru_ifa2_initiator`

Example of initiator node parameter configuration:

- **Type:** CINT example
- **Path:** `src/examples/dnx/instru/cint_ifa2.c`
- **Name**: `cint_ifa2_node_parameters_config`

Example of IFA 2.0 header parameter configuration:

- **Type:** CINT example
- **Path:** `src/examples/dnx/instru/cint_ifa2.c`
- **Name**:`cint_config_IFA2_header`

Example of initiator field configurations:

- **Type:** CINT example
- **Path:** `src/examples/dnx/instru/cint_ifa2.c`
- **Name**: `cint_field_IFA2_datapath_initiator_main`

50.3.6 Intermediate Node Configuration

The intermediate node detects IFA 2.0 packets by IP protocol ID. Current length and hop-limit are checked, and if the limits are not reached, IFA 2.0 metadata of the node is added. The current length and hop-limit fields are updated.

50.3.6.1 Configuration

A detailed example may be found in `cint_instru_ifa2_intermediate()`.

1. Global configuration:
   – Set node parameters, using `bcm_ifa_config_info_set` API.
   Example: `cint_ifa2_node_parameters_config`.
   – Set IFA 2.0 header parameters using
   `bcm_ifa_header_create`
   Example: `cint_config_IFA2_header`.
2. Set up field configurations, as in `cint_field_IFA2_datapath_intermediate_main`
   Configure iPMF1:
   – Qualifiers:
   IP-protocol is IFA 2.0
   – Actions:
   - Set Container for iPMF3
   - Set Tail Edit Profile = 6

   – Set TCAM Entry:
   On filter condition:
   - Action to signal iPMF3 to set INT
   - Action to set Tail Edit Profile = 6

   Configure iPMF3
   – Qualifier: Container is set.
   – Action: Set INT = 1 (Because INT cannot be set in iPMF1. It is done in iPMF3.)

50.3.6.2 Application Reference

Example of intermediate configuration steps:

- **Type:** CINT example
- **Path:** `src/examples/dnx/instru/cint_ifa2.c`
- **Name**: `cint_instru_ifa2_intermediate`

Example of initiator node parameter configuration:

- **Type:** CINT example
- **Path:** `src/examples/dnx/instru/cint_ifa2.c`
- **Name**: `cint_ifa2_node_parameters_config`

Example of IFA 2.0 header parameter configuration:

- **Type:** CINT example
- **Path:** `src/examples/dnx/instru/cint_ifa2.c`
- **Name**: `cint_config_IFA2_header`

Example of initiator field configurations:

- **Type:** CINT example
- **Path:** `src/examples/dnx/instru/cint_ifa2.c`
- **Name**: `cint_field_IFA2_datapath_intermediate_main`

50.3.7 Termination without Metadata Node Configuration

The termination node detects IFA 2.0 packets by IP protocol ID. IFA 2.0 packets are snooped to collector for analysis. Metadata is not added to current metadata stack. The original copy is stripped from the IFA 2.0 header, IFA 2.0 metadata header, and metadata stack. The packet is then forwarded to its destination.

50.3.7.1 Configuration

A detailed and accurate example may be found in `cint_instru_ifa2_termination()`.

1. Global configuration:
   – Set node parameters, using the `bcm_ifa_config_info_set` API.
   Example: `cint_ifa2_node_parameters_config`.
   – Set IFA 2.0 header parameters using `bcm_ifa_header_create`.
   Example: `cint_config_IFA2_header`.
2. Setup field configurations, as in `cint_field_IFA2_datapath_termination_main`
   Configure iPMF1:
   – Qualifiers:
   P-protocol is IFA 2.0
   – Actions:
   - Snoop packet
   - Set Container for iPMF3
   - Set Tail Edit Profile = 7

   – Set TCAM Entry:
   On filter condition:
   - Action to signal iPMF3 to set INT
   - Action to set Tail Edit Profile = 7

   Configure iPMF3
   – Qualifier: Container is set.
   – Action: Set INT = 1 (Because INT cannot be set in iPMF1. It is done in iPMF3.)

50.3.7.2 Application Reference

Example of intermediate configuration steps:

- **Type:** CINT example
- **Path:** `src/examples/dnx/instru/cint_ifa2.c`
- **Name**: `cint_instru_ifa2_termination`

Example of initiator node parameter configuration:

- **Type:** CINT example
- **Path:** `src/examples/dnx/instru/cint_ifa2.c`
- **Name**: `cint_ifa2_node_parameters_config`

Example of IFA 2.0 header parameter configuration:

- **Type:** CINT example
- **Path:** `src/examples/dnx/instru/cint_ifa2.c`
- **Name**: `cint_config_IFA2_header`

Example of initiator field configurations:

- **Type:** CINT example
- **Path:** `src/examples/dnx/instru/cint_ifa2.c`
- **Name**: `cint_field_IFA2_datapath_termination_main`

50.3.8 Termination with Metadata Node Configuration

The termination node detects IFA 2.0 packets by IP protocol ID. IFA 2.0 packets are snooped and encapsulated with the termination node's metadata. The IFA 2.0 packet is then forwarded to a collector for analysis. The original copy is stripped from the IFA 2.0 header, IFA 2.0 metadata header, and metadata stack. The packet is then forwarded to destination.

50.3.8.1 Configuration

A detailed example may be found in `cint_instru_ifa2_termination_with_md()`.

1. Global configuration:
   - Set node parameters, using `bcm_ifa_config_info_set` API. Example: `cint_ifa2_node_parameters_config`.
   - Set IFA 2.0 header parameters using `bcm_ifa_header_create`. Example: `cint_config_IFA2_header`.
2. Set up field configurations, as in `cint_field_IFA2_datapath_termination_with_md`.

   Configure iPMF1:
   - Qualifiers:

     IP-protocol is IFA 2.0
   - Actions:
     - Snoop packet
     - Set Container for iPMF3
     - Set Tail Edit Profile = 6
   - Set TCAM Entry:

     On filter condition:
     - Snoop packet
     - Action to signal iPMF3 to set INT
     - Action to set Tail Edit Profile = 7

   Configure iPMF3
   - Qualifier: Container is set.
   - Action: Set INT = 1 (Because INT cannot be set in iPMF1. It is done in iPMF3.)

3. Setup egress field configurations, as in `cint_instru_ifa2_egress_recycle`
   Configure ePMF:
   – Qualifiers:
   Snooped copy
   – Actions:
   - Set ACE to trap
   - Set Tail Edit Profile = 6
4. Setup trap configuration for recycling and forwarding to collector as in `cint_ifa2_termination_with_md_recycle_trap`
   – Configure ingress trap using type `bcmRxTrapUserDefine`
   – Configure egress trap using type
   – `bcmRxTrapEgTxIfaEgressMetadata`, and the ingress trap as `cpu_trap_gport`.

50.3.8.2 Application Reference:

Example of termination with metadata configuration steps:
- **Type:** CINT example
- **Path:** `src/examples/dnx/instru/cint_ifa2.c`
- **Name**: `cint_instru_ifa2_termination_with_md`

Example of termination with metadata node parameter configuration:
- **Type:** CINT example
- **Path:** `src/examples/dnx/instru/cint_ifa2.c`
- **Name**: `cint_ifa2_node_parameters_config`

Example of IFA 2.0 header parameter configuration:
- **Type:** CINT example
- **Path:** `src/examples/dnx/instru/cint_ifa2.c`
- **Name**: `cint_config_IFA2_header`

Example of termination with metadata ingress field configurations:
- **Type:** CINT example
- **Path:** `src/examples/dnx/instru/cint_ifa2.c`
- **Name**: `cint_field_IFA2_datapath_termination_with_md`

Example of termination with metadata egress field configurations:
- **Type:** CINT example
- **Path:** `src/examples/dnx/instru/cint_ifa2.c`
- **Name**: `cint_field_IFA2_egress_datapath_group` and `cint_field_IFA2_datapath_egress_entry_add`

Example of termination with metadata trap configurations:
- **Type:** CINT example
- **Path:** `src/examples/dnx/instru/cint_ifa2.c`
- **Name**: `cint_ifa2_termination_with_md_recycle_trap`

Chapter 51: Instrumentation – Elephant Flows

51.1 Introduction

An elephant flow is a data flow that consumes a disproportionately large amount of the total available bandwidth. Research on network traffic classification has shown that about 5% of the top data flows (that is, the largest data flows, known as elephant flows) may consume up to 95% of the traffic load. The impact of this type of flow on network behavior is a lack of resources for other flows, which leads to poor quality for the small data flows, known as mice flows.

To be able to react to and change switch behavior for handling elephant and mice flows, the first step is to detect and classify a flow as an elephant or mice flow. The second step is to set proper actions (for example, create separate queues for elephant flows to keep mice flow latency low).

After the flow is classified as an elephant flow, a report is generated (the packet is snooped to the control plane). After a flow is reported to the CPU, its state is changed, and no more reports will be sent to the CPU for this specific flow to avoid flooding the CPU with the same flow reports. CPU handling for reacting upon receiving the report is up to user implementation.

NOTE: When a flow is detected as Elephant, a report is sent to the CPU, and its state is changed internally to avoid flooding the CPU with the same flow report.

There is a built-in race between changing the internal state and packets coming from the same flow, so there might be several Elephant reports for the same flow.

51.2 Reference

For more information, refer to the following

- BCM88690 *Packet Processing Architecture* (88690-DG2xx), Elephant Trap section.
- Chapter 11, Field Processor.

51.3 Application Configuration Checklist

APIs are available to configure the following settings related to elephant flows:

- Set the port trace probability.
- Create a mirroring and snooping action to the CPU.
- Create a field-group for identifying a flow (5-tuple or other criteria).
- Create an age flush profile and attach the Field Group to it.
- Set flush machine rules and actions for reporting an elephant flow.
- Set the age machine scan period.

51.4 Port Trace Probability

To perform statistical operations in PMF, one of the qualifiers should be the `trace_probability` bit.

For more information, see Section 48.4, Trace Probability.

51.5 Packet Mirror and Snoop

When a flow is classified as an elephant flow, a report should be sent to the control plane to react. The report is actually the original packet of that flow sent to a dedicated CPU port. For more information and packet mirroring and snooping, refer to the Sniff chapter in the *Traffic Manager Programming Guide* (88690-PG2xx).

51.6 Data Flow Identification

Data-flow identification is usually done by:

- 5-tuple {Src_IP,Dst_IP,Protocol,L4_Src_Port,L4_Dst_port} is used to create a 16b CRC (hash) signature. Invoke `bcm_field_context_hash_create()` in the iPMF1 stage.

  For more information, see Section 11.18, Hash.
- The 16b signature can be used as part of the key in the database. Based on the result, a snoop or learn payload is set. The Field Group should be added to the iPMF2 stage.
  - `bcmFieldActionSmallExemLearn` should be used to set the payload to be learned and should include:
    - State bits – None, mice, elephant, or elephant reported
    - Snoop command – If the state will be changed to elephant
    - Age flush profile that was created by `bcm_field_flush_profile_create()`

51.6.1 SOC Properties

```
pmf_sexem3_stage.BCM8880X=IPMF2
```

51.6.2 Configuration Flow

For configuration flow information, see Section 11.18, Hash and Section 11.20, Exact Match Lookup.

51.6.3 Shell Commands

Use field diagnostics. See Chapter 11, Field Processor.

51.7 Age Flush Profile

To configure the age flush profile for EXEM FG, invoke the following two APIs:

- `bcm_field_flush_profile_attach()`
- `call bcm_switch_control_indexed_set()` – Indicates where in `bcmFieldActionSmallExemLearn` the age profile is located.

  The action size is 32b, so the offset should be in this range.

51.7.1 Configuration Flow

Create an age flush profile:

```
uint32 flush_profile_id;
bcm_field_flush_profile_info_t flush_profile_info;

bcm_field_flush_profile_info_t_init(&flush_profile_info);

flush_profile_info.high_threshold_value = <ELEPHANT THRESHOLD>;
flush_profile_info.low_threshold_value = <MICE THRESHOLD>;
flush_profile_info.increment_value = <INCREMENT VALUE>;
flush_profile_info.decrement_value = <DECREMENT VALUE>;
flush_profile_info.maximal_value = <MAXIMAL VALUE>;
flush_profile_info.out_value = <AGE OUT VALUE>;
flush_profile_info.init_value = <INITIAL VALUE>;

bcm_field_flush_profile_create(unit,0, &flush_profile_info, &flush_profile_id);
```

NOTE: The flag `BCM_FIELD_FLAG_WITH_ID` can be used for allocating a specific profile ID.

Attach a field-group to an age flush profile:

```
bcm_field_flush_profile_attach(unit,0, fg_id, flush_profile_id);
```

NOTE:

- The age flush profile must be created before it can be attached.
- The field-group must be created before it can be attached to an age flush profile.
- The age flush profile can be mapped to only one field group.

Setting Flush-profile payload offset in table payload:

```
bcm_switch_control_key_t control_key;
bcm_switch_control_info_t control_info;

control_key.type = bcmSwitchExemFlushProfilePayloadOffset;
control_key.index = database <0 = EXEM-3>
control_info.value = <AGE ACTION OFFSET>;

bcm_switch_control_indexed_set(unit, control_key, control_info);
```

Destroying an age flush profile:

```
bcm_field_flush_profile_destroy(unit,0, flush_profile_id);
```

NOTE: The age flush profile HW is set to *ALL ZERO*, thus the aging mechanism and the elephant mechanism are disabled for this profile.

51.8 Flush Machine Rules and Actions

For elephant flow detection, a field in the EXEM payload is configured to hold the flow state.

Flow states are as follows:
- MOUSE = 1
- ELEPHANT = 2
- ELEPHANT REPORTED = 3

When an entry is first triggered to start learning, its state is set to MOUSE.

The following rules and actions are configured for supporting elephant flow detection:
- MOUSE to ELEPHANT rule:
  - Criteria: state == MOUSE and hitbit == 1
  - Action: Transplant command – State is changed to ELEPHANT.
- ELEPHANT to MOUSE rule:
  - Criteria: state == ELEPHANT and hitbit == 0
  - Action: Transplant command – State is changed to MOUSE.
- ELEPHANT REPORTED to MOUSE rule:
  - Criteria: state == ELEPHANT REPORTED and hitbit == 0
  - Action: Transplant command – State is changed to MOUSE.

When an entry is in the ELEPHANT state, on the next hit, the PMF is configured to perform the following two actions:
- Trigger the LBP transplant command for changing the entry state to ELEPHANT REPORTED.
- Snoop the packet to the CPU.

51.8.1 Configuration Flow

Add a flush rule and action:

```
bcm_field_flush_entry_info_t flush_entry_info;
bcm_field_flush_entry_info_t_init(&flush_entry_info);

/* fill key_info with all “rule” information */
/* fill payload_info with all “action” information */

bcm_field_flush_entry_add(unit, 0, fg_id, entry_id, &flush_entry_info);
```

51.9 Age Scan Period

During every age scan period, the EXEM is scanned, and each entry’s age counter is updated based on its hitbit value (incremented for a hit and decremented for a miss).

The value of the age scan period must be set according to its application preference.

51.9.1 SOC Properties

None

51.9.2 Configuration Flow

Setting age scan period:

```
bcm_switch_control_key_t control_key;
bcm_switch_control_info_t control_info;

control_key.type = bcmSwitchExemScanPeriod;
control_key.index = database>
control_info.value = <AGE SCAN PERIOD, in uSec>;

bcm_switch_control_indexed_set(unit, control_key, control_info);
```

NOTE: EXEM_4 is not supported. Only EXEM_3 can be used for Elephant Flow.

51.9.3 Application Reference

Elephant flow application example:

- **Type**: CINT reference
- **Path:** `$SDK/src/examples/dnx/elephant_flow/cint_elephant_flow.c`
- **Function** `cint_elephant_flow_set_age_period()`.

51.10 API Descriptions

For a full description of the following field flush APIs used for elephant flow detection implementation, see Chapter 11, Field Processor.

API Name	Highlights
`bcm_switch_control_indexed_set()`	`bcmSwitcinstruhExemFlushProfilePayloadOffset`: Set the EXEM flush profile payload offset.
`bcm_switch_control_indexed_set()`	`bcmSwitchExemScanPeriod`: Set the EXEM age scan period.
`bcm_instru_gport_control_set()`	`bcmInstruGportControlTraceProbability`: Set a port's trace probability.

Chapter 52: Instrumentation – IPFIX

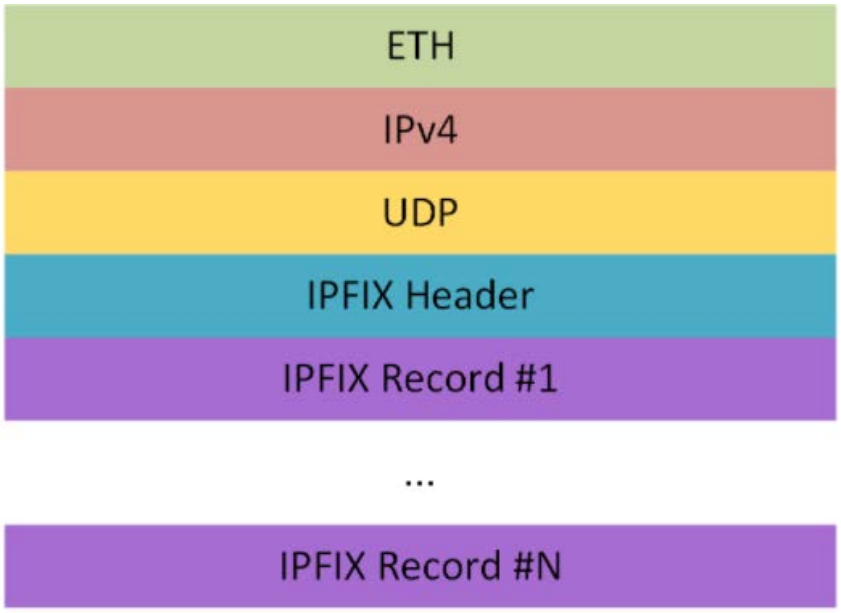
52.1 Introduction

Traffic on a data network consists of flows passing through network elements. Network administrators often require access to information about these flows.

IP Flow Information Export (IPFIX) describes a way to collect the flow information and communicate the information to a collection point. The application samples the traffic and collects information about the sampled packets (IPFIX record). Collected information is sent to the eventor and aggregated. The records are sent (exported) to a collector entity after a configurable number of records has been gathered.

The following figure shows the structure of the packet exported to the collector.

Figure 47: IPFIX Packet Structure



The following figure shows the format of the IPFIX header.

Figure 48: IPFIX Header

IPFIX Version = 0xA	IPFIX Length = 16+44*N
IPFIX Export Time (Sec)	
IPFIX Sequence Number	
IPFIX Observation Domain	

The IPFIX header fields are as follows:

- IPFIX Version [16b] = 0xA.
- IPFIX Length [16b] – Length of the IPFIX export packet, including the IPFIX header and the sample records.
- IPFIX Export Time [32b] – IPFIX export packet's transmission time, in seconds.
- IPFIX Sequence Number [32b] – Sequence number of the IPFIX export packet.
- IPFIX Observation Domain [32b]

The IPFIX record fields are as follows:

- IPFIX Template ID [16b].
- Sample Length [16b] – Length of the sample data = 40.
- Version [4b] – IP Version = 4.
- TOS[8b] – IPv4 TOS field.
- Protocol [8b] – IPv4 protocol field.
- SIP [32b] – IPv4 SIP field.
- DIP [32b] – IPv4 DIP field.
- L4 Src port [16b] – For TCP and UDP protocols, contains the source port from Layer 4. Set to 0 for other Layer 4 protocols.
- L4 Dest port [16b] – For TCP and UDP protocols, contains the destination port from Layer 4. Set to 0 for other Layer 4 protocols.
- TCP Flags [12b] – For TCP protocol, contains the TCP flags field. Set to 0 for other Layer 4 protocols.
- ICMP Type [8b] – For ICMP protocol, contains the ICMP type field. Set to 0 for other Layer 4 protocols.
- ICMP Code [8b] – For ICMP protocol, contains the ICMP code field. Set to 0 for other Layer 4 protocols.
- IPFIX Ingress VRF [32b] – Ingress VRF-ID associated with the source port.
- IPFIX Ingress Interface [16b] – Ingress interface associated with the source port.
- IPFIX Egress VRF [32b] – Egress VRF-ID associated with the destination port.
- IPFIX Egress Interface [16b] – Egress interface associated with the destination port.
- Setup Time [32b] – IPFIX setup time.

52.2 References

- IETF RFC-7011: https://datatracker.ietf.org/doc/html/rfc7011
- IPFIX details: https://www.iana.org/assignments/ipfix/ipfix.xhtml

52.3 Application Configuration Checklist

The IPFIX feature includes the following configuration options:

- Global IPFIX configuration:
  - Create an eventor port and enable IPFIX on that port.
  - Allocate the counter resources.
  - Configure global IPFIX parameters.
- Set the required eventor builders and contexts for sample aggregation and transmission.
- IPFIX configuration:
  - Egress: Create an IPFIX encap object.
  - Ingress: Snoop-stat sampling object that points to IPFIX encap.
- Field processor configuration for IPFIX:
  - Create field groups to match the IPFIX application.
  - Add a field entry per IPFIX.

52.4 Global Configuration

The global configurations for IPFIX use the following configurations:

- Setup time – Value to stamp on the sampled metadata.
- Template ID – Value to stamp on the sampled metadata.
- Eventor Header – ID of the context the eventor uses.
- Export time – Value to be stamped on the IPFIX header. The time stamp should be constantly updated by software. The updated value is stamped on the packet.
- Observation Domain – Value to be stamped on the IPFIX header.
- Stat counter configuration – Resource allocation for the counter to be used as sequence number, and stamped on the IPFIX header.

52.4.1 SOC Properties

The eventor requires a logical port for TX/RX Eventor:

- Create a logical port by setting the `ucode_port_XXX=EVENTOR:core_0.YYY` SOC property
- Use the following SOC properties to set the header type for the eventor port:
  - `tm_port_header_type_in_XXX=INJECTED_2`
  - `tm_port_header_type_out_XXX=ETH`

52.4.2 Configuration Flow

To add the counter resources, see Chapter 15, PP Statistics Generation and the CINT reference in Section 52.4.3, Application Reference.

The configuration flow includes the following actions:

- To configure the export timestamp, initialize a thread that constantly checks the time (in seconds), and then call the following API to update:
  ```
  bcm_instru_control_set(unit, 0, bcmInstruControlIpFixTxTime, export_time);
  ```
- To configure the setup time, call:
  ```
  bcm_instru_control_set(unit, 0, bcmInstruControlIpFixSystemUpTime,  uptime);
  ```
- To configure Template ID, call:
  ```
  bcm_instru_control_set(unit, 0, bcmInstruControlIpFixTemplateId, TEMPLATE_ID);
  ```
- To configure Eventor header, call:
  ```
  bcm_instru_control_set(unit, 0, bcmInstruControlIpFixEventorId, eventor_id);
  ```
- To configure observation domain, call:
  ```
  bcm_instru_control_set(unit, 0, bcmInstruControlIpFixObservationDomain,
  observation_domain);
  ```

52.4.3 Application Reference

Example of counters usage:

- **Type:** CINT example
- **Path:** `src/examples/dnx/oam/cint_oam_basic.c`
- **Function:** `set_counter_resource`

Example of IPFIX global register usage:

- **Type:** CINT example
- **Path:** `src/examples/dnx/ipfix/cint_ipfix.c`
- **Function:** `cint_ipfix_registers_init`

52.5 Set Eventor Context and Builder

The eventor aggregates a number of samples to be sent on a single IPFIX datagram to the collector.

The eventor configuration consists of RX context and TX builders. Context configuration determines the used banks and their sizes. Builder configuration determines the number of samples to aggregate and to send on a single datagram and the predefined header of the IPFIX datagram.

52.5.1 Configuration Flow

1. Set the eventor context:
   - `bcm_instru_eventor_context_conf_t context_conf = {0};`
   - `context_conf.bank1 = <bank_idx>`
   - `context_conf.bank2 = <bank_idx>`
   - `context_conf.buffer_size = <size in 4 byte words>`
   - `context_conf.buffer1_start = <offset in 4 bytes words in 1st buffer>`
   - `context_conf.buffer2_start = <offset in 4 bytes words in 2nd buffer>`
   - `uint32 flags = 0;`
   - `int context = <context_idx>`
   - `bcm_instru_eventor_context_set(unit, flags, context, bcmEventorContextTypeRx, &context_conf);`
2. Set the eventor builder:
   - `bcm_instru_eventor_builder_conf_t builder_conf = {0};`
   - `builder_conf.flags = 0;`
   - `uint8 header_data[128] = <raw data: predefined header, same as bcm_tx format>`
   - `builder_conf.thresh_size = <buffer_size_thr>; /* When this amount of 4 byte words is reached, generate a packet */`
   - `builder_conf.thresh_time = BCM_INSTRU_EVENTOR_TIMEOUT_NONE;`
   - `builder_conf.header_data = header_data; /* predefined header */`
   - `builder_conf.header_length = header_length; /* The length, in bytes, of the header to use in generated packets */`
   - `uint32 flags = 0;`
   - `int builder = <builder_idx>`
   - `bcm_instru_eventor_builder_set(unit, flags, builder, &builder_conf);`

The header data format is according to incoming header type (for example, `INJECTED_2`):

- PTCH2 header (`Parser_Program_Control` is 1, In-PP-port don't care)
- ETH header
- IPv4 UDP-Tunnel header (UDPoIPv4oETH)
  The UDP checksum is not calculated. Its value is set by the user in `header_data`.
- IPFIX header

52.5.2 Application Reference

Example of eventor configuration for IPFIX:

- **Type:** CINT example
- **Path:** `src/examples/dnx/ipfix/cint_ipfix.c`
- **Functions:** `cint_ipfix_eventor_set` and `cint_ipfix_eventor_activate`

52.6 IPFIX Configuration – Egress

A UDP tunnel is done as part of the eventor builder header data.

Egress configuration for IPFIX selects the correct egress contexts and builds a sample datagram that is sent to the eventor for aggregation.The sample datagram is sent to the eventor with internal information that includes the eventor RX context.

IPFIX interface information to be stamped on the sample-datagram must be set.

52.6.1 Configuration Flow

Create IPFIX Encap object by calling `bcm_instru_ipfix_encap_create(unit, &ipfix_encap_info);`

The IPFIX Encapsulation object creates the IPFIX header datagram-sample, and sets the eventor header with Eventor internal information (for example, Eventor RX context).

The input and output interface information to stamp on the sample-datagram is set for each of its sampled packet source and destination. The sampled packet source is a system-port (the system-port may be part of a LAG), while the sampled packet destination can be a system-port (but *not* a system-port that is part of LAG), LAG, multicast group, or flow ID.

Interface and VRF IDs that are stamped on the sample-datagram are configured using the `ipfix_interface` object. Input and output interfaces must be set separately. This requires the following two API calls:

For input:

```
ipfix_iterface.flags = BCM_INSTRU_IPFIX_INTERFACE_INPUT;
ipfix_iterface.port = gport;
ipfix_iterface.interface = intf_in;
ipfix_iterface.vrf_id = vrf;
bcm_instru_ipfix_interface_add(unit, &ipfix_iterface);
```

For output:

```
ipfix_iterface.flags = BCM_INSTRU_IPFIX_INTERFACE_OUTPUT;
output_interface = out_port;
ipfix_iterface.port = output_interface;
ipfix_iterface.interface = intf_out;
ipfix_iterface.vrf_id = vrf;
bcm_instru_ipfix_interface_add(unit, &ipfix_iterface);
```

Enable IPFIX on eventor port using:

```
bcm_port_control_set(unit, eventor_port,bcmPortControlIpFixRxEnable,1);
```

Perform egress field configuration to allow the correct context selection in egress. See Section 52.8, Field Configuration.

52.6.2 Application Reference

Example of IPFIX interface usage:

- **Type:** CINT example
- **Path:** `src/examples/dnx/ipfix/cint_ipfix.c`
- **Function:** `cint_ipfix_egress_create`

Example of IPFIX port enable:

- **Type:** CINT example
- **Path:** `src/examples/dnx/ipfix/cint_ipfix.c`
- **Function:** `cint_ipfix_egress_create`

52.7 IPFIX Configuration – Ingress

Ingress configuration includes the snoop stat sampling and ingress field configuration.

The Snoop-stat sampling object serves several purposes in the IPFIX application:

- It mirrors the packet for sampling in some probability.
- It builds new system headers with the following information:
  - New destination port (to the collector).
  - Keep the original encap ID.

52.7.1 Configuration Flow

Create mirroring by using the `bcm_mirror_destination_create(unit, &mirror_dest)` API.

The following are the required flags:

- `BCM_MIRROR_DEST_IS_STAT_SAMPLE` to set snooped packet's type to statistical sampling.
- `BCM_MIRROR_DEST_TUNNEL_WITH_ENCAP_ID` to set the `encap_id` on the packet's header.

The parameters are as follows:

- `mirror_dest.flags` = `BCM_MIRROR_DEST_IS_STAT_SAMPLE` or `BCM_MIRROR_DEST_TUNNEL_WITH_ENCAP_ID`
- `mirror_dest.encap_id = ipfix_encap_id`
- `mirror_dest.packet_copy_size = 256` – This is a fixed value.
- `mirror_dest.sample_rate_dividend = sample_rate_dividend` – The valid values are 0 to disable or 1 to enable.
- `mirror_dest.sample_rate_divisor = sample_rate_divisor`
- `mirror_dest.gport = eventor_port`

For information about ingress field configuration for sampling and correct IPFIX datagram stamping, see Section 52.8, Field Configuration.

52.7.2 Application Reference

Example of IPFIX mirroring usage:

- **Type:** CINT example
- **Path:** `src/examples/dnx/ipfix/cint_ipfix_field.c`
- **Function:** `cint_ipfix_mirror_create`

52.8 Field Configuration

The ingress PMF samples the packet flow, prepares port information for stamping, and exports the sequence number increment. Two groups are attached to the same iPMF context: one group is for sampling the packet, and another group is for incrementing the counter.

The egress PMF selects the correct context.

52.8.1 Configuration Flow

The flow for iPMF1 is as follows:

- Group 1 for received packet:
  - Qualifier:
    - 5-tupple
    - `in_port` == `eventor_port` (qualifier for preventing snooping of an IPFIX export packet coming from the eventor port)
  - Action:
    - Set the packet type to statistical sampling
    - Set the container for iPMF3
  - Entries:
    - On sampling condition, set the statistical sampling and container for iPMF3
    - On `in_port` == `eventor_port`, do nothing to prevent export packets that come from the eventor to be snooped
- Group 2 for exported packet:
  - Qualifier:
    - `in_port` == `eventor_port`
    - Layer 3 == IPv4
    - UDP dest port == IPFIX
    - Source port == eventor
  - Action:
    - Set the counter for the IPFIX sequence number
    - Set the IPFIX sub-type
  - Entries:

    Under the following conditions, set the counter ID and `sub_type`:
    - `in_port` == `eventor_port`
    - Layer3 == IPv4
    - UDP dest port == IPFIX
    - Source port == eventor

- iPMF3
  Group 1
  - Qualifier – Container from iPMF1
  - Action – Set the source and destination ports on UDH (for the egress for IPFIX interface information stamping)
- ePMF
  Group 1
  - Qualifier – Snooped copy
  - Action – Set `IpfixContext` as an ACE context value
  - Entry – On a snooped copy, set the ACE context value

52.8.2 Application Reference

Example of IPFIX field usage iPMF1 Group 1:

- **Type:** CINT example
- **Path:** `src/examples/dnx/ipfix/cint_ipfix_field.c`
- **Functions:**
  - `cint_ipfix_field_main`
  - `cint_ipfix_field_entry_add`
  - `cint_ipfix_field_entry_add_eventor_port`

Example of IPFIX field usage iPMF1 Group 2:

- **Type:** cint example
- **Path:** `src/examples/dnx/ipfix/cint_ipfix_field.c`
- **Functions:**
  - `cint_ipfix_field_main`
  - `cint_ipfix_field_export_packet`
  - `cint_ipfix_field_export_quals_set`

Example of IPFIX field usage iPMF3:

- **Type:** CINT example
- **Path:** `src/examples/dnx/ipfix/cint_ipfix_field.c`
- **Functions:**
  - `cint_ipfix_field_main`
  - `cint_ipfix_field_entry_add`
  - `cint_ipfix_field_entry_add_eventor_port`

Example of IPFIX field usage ePMF:

- **Type:** CINT example
- **Path:** `src/examples/dnx/ipfix/cint_ipfix_field.c`
- **Functions**:
  - `cint_ipfix_field_egress_group`
  - `cint_ipfix_field_egress_entry_add`

52.9 API Description

API Name	Highlights
`bcm_instru_ipfix_encap_create(int unit, bcm_instru_ipfix_encap_info_t * ipfix_encap_info)`	Create an IPFIX OutLIF entity.
`bcm_instru_ipfix_encap_get(int unit, bcm_instru_ipfix_encap_info_t * ipfix_encap_info)`	Get an IPFIX entity, based on the encap ID (IPFIX global LIF) specified.
`bcm_instru_ipfix_encap_delete(int unit, bcm_instru_ipfix_encap_info_t * ipfix_encap_info)`	Deletes an IPFIX OutLIF entity.
`bcm_dnx_instru_ipfix_encap_traverse(int unit, bcm_instru_ipfix_encap_traverse_cb cb, void *user_data)`	Traverse all IPFIX entities.
`bcm_dnx_instru_ipfix_interface_add(int unit, bcm_instru_ipfix_interface_info_t * ipfix_interface_info)`	Add an IPFIX interface.
`bcm_dnx_instru_ipfix_interface_get(int unit, bcm_instru_ipfix_interface_info_t * ipfix_interface_info)`	Get an IPFIX interface.
`bcm_dnx_instru_ipfix_interface_remove(int unit, bcm_instru_ipfix_interface_info_t * ipfix_interface_info)`	Remove an IPFIX interface.
`bcm_dnx_instru_ipfix_interface_traverse(int unit, bcm_instru_ipfix_interface_traverse_info_t * ipfix_interface_traverse_info, bcm_instru_ipfix_interface_traverse_cb cb, void *user_data)`	Traverse all IPFIX interfaces.
`bcm_dnx_port_control_set(int unit,bcm_port_t port,bcm_port_control_t type,int value)`	Set various features at the port level. To be used with the `bcmPortControlIpFixRxEnable` type.
`bcm_dnx_port_control_get(int unit,bcm_port_t port,bcm_port_control_t type,int *value_p)`	Get various features at the port level. To be used with the `bcmPortControlIpFixRxEnable` type.
`bcm_instru_control_set(int unit, uint32 flags, bcm_instru_control_t type, int arg);`	Set various IPFIX parameters. To be used with `bcmInstruControlIpFixSystemUpTime`, `bcmInstruControlIpFixEventorId`, `bcmInstruControlIpFixTemplateId`, `bcmInstruControlIpFixTxTime`, and `bcmInstruControlIpFixObservationDomain` types.
`bcm_instru_control_set(int unit, uint32 flags, bcm_instru_control_t type, int arg);`	Set various IPFIX parameters. To be used with `bcmInstruControlIpFixSystemUpTime`, `bcmInstruControlIpFixEventorId`, `bcmInstruControlIpFixTemplateId`, `bcmInstruControlIpFixTxTime`, and `bcmInstruControlIpFixObservationDomain` types
`bcm_instru_eventor_context_set(int unit, uint32 flags, bcm_eventor_context_id_t context, bcm_eventor_context_type_t context_type, bcm_instru_eventor_context_conf_t *conf);`	Set the eventor context.
`bcm_instru_eventor_context_get(int unit, uint32 flags, bcm_eventor_context_id_t context, bcm_eventor_context_type_t context_type, bcm_instru_eventor_context_conf_t *out_conf);`	Get the eventor context.
`bcm_instru_eventor_builder_set(int unit, uint32 flags, bcm_eventor_builder_id_t builder, bcm_instru_eventor_builder_conf_t *conf);`	Set the configuration of the given builder.

API Name	Highlights
`bcm_instru_eventor_builder_get(int unit, uint32 flags, bcm_eventor_builder_id_t builder, bcm_instru_eventor_builder_conf_t *out_conf);`	Get the configuration of the given builder.
`bcm_instru_eventor_active_set(int unit, uint32 flags, int active)`	Start or stop the eventor.

Appendix A: Device Family Differences

Table 97: Device Family Differences by Feature

Topic	Parameter	BCM88690\ BCM88395 (Jericho2\QMX2)	BCM88670\ BCM88375 (Jericho\QMX)	BCM88800 (J2C)	BCM88480 (Q2A)	BCM88830 (J2X)
Global	Number of device cores	2	2	1	1	1
Ports	OTMH support	X	✓	✓	✓	X
Ethernet-Bridge	Number of TPIDs	7 global TPIDs Each port can assign each of the 7 TPIDs	4 global TPIDs Each port can assign up to 2 TPIDs	7 global TPIDs Each port can assign each of the 7 TPIDs	7 global TPIDs Each port can assign each of the 7 TPIDs	7 global TPIDs Each port can assign each of the 7 TPIDs.
	External (KBP) TCAM lookup supported	✓	✓	✓	✓	X
Port Extender Channelized over Ethernet (COE)	Feature support	X	✓	✓	✓	✓

Appendix B: General Control APIs for Packet Processing

This appendix provides information about general control APIs in the SDK. Each control API includes a short highlight and a reference for where to find more information.

B.1 bcm_port_control_set

Description: General properties per port control. The following table summarizes all supported control packet-processing types.

Table 98: bcm_port_control_set

Port Control Type (bcm_port_control_t)	Port/LIF Type and In/Out	Highlights	PG Document Reference Section
bcmPortControlMplsFRREnable	InPort	Enable/Disable FRR (Fast ReRoute) for MPLS	Chapter 23, MPLS Label Switch Router
bcmPortControlDiscardEgress	OutPort	Configure port to drop all packets	Chapter 2, Ports and Generalized Ports
bcmPortControlEgressFilterDisable	OutPort	Disable egress filters for Unknown type packets Available for Interop mode only (BCM8867X system headers).	Chapter 5, Interoperability with Legacy Devices
bcmPortControlOuterPolicerRemark	InLIF	Set DP profile on LIF for IVE outer PCP mapping	Chapter 7, Logical Interface (LIF) Management
bcmPortControlInnerPolicerRemark	InLIF	Set DP profile on LIF for IVE inner PCP mapping	Chapter 7, Logical Interface (LIF) Management
bcmPortControlExtenderType	InPort	Provide Port extender type mode.	Chapter 40, Port Extender Channelization over Ethernet
bcmPortControlBridge	OutPort, InLIF	Enable Same interface filter	Chapter 21, Ethernet Bridge
bcmPortControlLogicalInterfaceSameFilter	InLIF	Set In LIF same interface mode.	Chapter 7, Logical Interface (LIF) Management
bcmPortControlPreserveDscpIngress	InLIF	Set In LIF preserve DSCP mode	Chapter 10, QoS
bcmPortControlOamDefaultProfile	InLIF	Configure OAM default profile	Chapter 7, Logical Interface (LIF) Management
bcmPortControlForwardNetworkGroup	OutPort	Enable/Disable Split Horizon	Chapter 20, L2 Generalized Bridging Model
bcmPortControl1588P2PDelay	InPort	Set 1588 ptp delay per port	Chapter 36, IEEE 1588v2 Precision Time Protocol
bcmPortControlIngressQosModelRemark	InPort	Set ingress QOS Remark model Configure Remark strength	Chapter 10, QoS
bcmPortControlIngressQosModelPhb	InPort	Set ingress QOS PHB model Configure PHB strength	Chapter 10, QoS
bcmPortControlIngressQosModelTtl	InPort	Set ingress QOS TTL model Configure TTL strength	Chapter 10, QoS
bcmPortControlOverlayRecycle	InPort	Set Port Properties to recognize RCH header	Chapter 18, Drop-and-Continue

Table 98: bcm_port_control_set (Continued)

Port Control Type (bcm_port_control_t)	Port/LIF Type and In/Out	Highlights	PG Document Reference Section
bcmPortControlExtenderEnable	In and Out Port	Enable Port to be extender (COE, 802.1BR and other Extender types)	Chapter 40, Port Extender Channelization over Ethernet
bcmPortControlDistributeLearnMessageMode	OutPort	Configure OLP messages to handle correctly BCM8867X system headers, Interop mode	Chapter 5, Interoperability with Legacy Devices
bcmPortControlFirstHeaderSize	InPort	Skip first header size before injected header according to the configured value	Chapter 2, Ports and Generalized Ports
bcmPortControlFirstHeaderSizeAfterInjected	InPort	Skip first header size after injected header according to the configured value	Chapter 2, Ports and Generalized Ports
bcmPortControlPONEnable	InPort	Enable PON application	Chapter 43, PON Application
bcmPortControlRecycleApp	InPort	Enable extended termination, additional encapsulations, additional terminations, reprocessing with PTCH1Plus	Chapter 18, Drop-and-Continue
bcmPortControlSystemPortInjectedMap	InPort	Enable mapping of system port to local-port to be used for injected scenarios (PTCH1)	Chapter 2, Ports and Generalized Ports
bcmPortControlEgressSystemPortInjectedMap	InPort	Enable mapping of system port to local-port to be used for injected scenarios (PTCH1). Relevant for recycle, in which the mapping will be made at the egress first pass. Valid only if bcmSwitchInjectedHeaderMode is True.	Chapter 2, Ports and Generalized Ports
bcmPortControlSaMulticastEnable	InPort	Enable or disable an SA multicast trap on the ingress port	Chapter 12, Traps

B.2 bcm_switch_control_port_set

Description: General properties per port control. The following table summarizes all supported switch port control packet-processing types.

Table 99: bcm_switch_control_port_set

Global Control Type (bcm_switch_control_t)	Port/LIF Type and In/Out	Highlights	PG Document Reference Section
bcmSwitchPrependTagEnable	OutPort	Enable/Disable prepend tag for port	Chapter 40, Port Extender Channelization over Ethernet
bcmSwitchTagPcpDeiSrc	OutPort	Set PCP-DEI source for 802.1BR	Chapter 40, Port Extender Channelization over Ethernet

Table 99: bcm_switch_control_port_set (Continued)

Global Control Type (bcm_switch_control_t)	Port/LIF Type and In/Out	Highlights	PG Document Reference Section
bcmSwitchPortHeaderType	Port (Both In&Out)	Configure general Port-attributes according to the port header type that is set for both directions.	Chapter 2, Ports and Generalized Ports
bcmSwitchHashLayersDisable	InPort	Remove protocols from participating in the hash for a given port	Chapter 13, Load Balancing

B.3 bcm_switch_control_indexed_set

Description: General properties per switch control. The following table summarizes all supported switch control packet-processing types:

Table 100: bcm_switch_control_indexed_set

Global Control type (bcm_switch_control_t)	Highlights	PG Document Reference Section
bcmSwitchLinkLayerMtuFilter	Map layer type to compressed layer type for MTU trapping. Indicates the layer type of the LIF/RIF that will be supplied to the MTU configuration structure. Relevant only for MTU per LIF/RIF.	Chapter 12, Traps
bcmSwitchForwardingLayerMtuFilter	Enable or disable MTU check for ETPP IPv4 forwarding context	Chapter 12, Traps
bcmSwitchHashSeed	Set CRC function seed	System properties and General notes
bcmSwitchLayerRecordModeSelection	Select the layer record bits for the key generation	Chapter 13, Load Balancing
bcmSwitchL3LpmHitbitEnable	Enable or disable raising a hit bit for any of the following LPM routing tables	Chapter 22, IP Router
bcmSwitchModuleVerifyEnable	Enable and disable the verification of certain module by switch module id	N/A
bcmSwitchModuleErrorRecoveryEnable	Enable or disable error recovery per module	N/A
bcmSwitchMplsSpeculativeNibbleMap	Change the default nibble speculation	Chapter 13, Load Balancing
bcmSwitchNdpMyIp1	Configure Ip1 of My-NDP protocol trap	Chapter 12, Traps
bcmSwitchNdpMyIp2	Configure Ip2 of My-NDP protocol trap	Chapter 12, Traps
bcmSwitchL3TunnelCollapseDisable	Disable/enable GTP-U collapse	Chapter 47, GPRS Tunneling Protocol
bcmSwitchMplsSpecialLabelAutoTerminate Disable	Set MPLS special label to non special one	Disable auto termination for special MPLS label (Value range 0 to 15) by tunnel termination. After this operation, the label will be treated as a general one. It could be terminated by MPLS Label lookup in relevant DB.

Table 100: bcm_switch_control_indexed_set (Continued)

Global Control type (bcm_switch_control_t)	Highlights	PG Document Reference Section
bcmSwitchEcmpEmptyEgressId	Configure a drop FEC per hierarchy.	Section 22.10, L3 Egress: ECMP Object

B.4 bcm_switch_control_indexed_port_set

Description: General properties per port. The following table summarizes all supported switch control **Packet-Processing** types.

Table 101: bcm_switch_control_indexed_port_set

Global Control Type (bcm_switch_control_t)	Highlights	PG Document Reference Section
bcmSwitchPortHeaderType	Set port`s header type according to BCM_SWITCH_PORT_HEADER_TYPE_XXX	Chapter 2, Ports and Generalized Ports

B.5 bcm_switch_control_set

Description: General properties per device. The following table summarizes all supported switch control Packet-Processing types.

Table 102: bcm_switch_control_set

Global Control Type (bcm_switch_control_t)	Highlights	PG Document Reference Section
bcmSwitchECMPHashConfig	Set ECMP-1 CRC function to a client	Chapter 13, Load Balancing
bcmSwitchECMPSecondHierHashConfig	Set ECMP-2 CRC function to a client	Chapter 13, Load Balancing
bcmSwitchECMPThirdHierHashConfig	Set ECMP-3 CRC function to a client	Chapter 13, Load Balancing
bcmSwitchTrunkHashConfig	Set LAG CRC function to a client	Chapter 13, Load Balancing
bcmSwitchNwkHashConfig	Set Network CRC function to a client	Chapter 13, Load Balancing
bcmSwitchMplsStack0HashSeed	Set the seeds for the MPLS stack hashing	Chapter 13, Load Balancing
bcmSwitchParserHashSeed	Set the seeds for the parsing hashing stage	Chapter 13, Load Balancing
bcmSwitchTraverseMode	Update the traverse mode from regular to bulk and back. Add rules and clear rules in bulk mode.	Chapter 21, Ethernet Bridge
bcmSwitchL2LearnMode	Configure learn mode	Chapter 21, Ethernet Bridge
bcmSwitchMactAgeRefreshMode	Configure the age refresh mode to be triggered by the DA hit, SA hit or a combination between them	Chapter 21, Ethernet Bridge
bcmSwitchL2AgeScan	Trigger a cycle of the age machine. The age of the MAC entries is reduced in the same way as if a meta cycle time had passed. It can be called only when the automatic aging is disabled.	Chapter 21, Ethernet Bridge
bcmSwitchL2LearnLimitToCpu	Enable and disable the limit check for MACT learning events	Chapter 21, Ethernet Bridge
bcmSwitchEtagEthertype	Set E-tag TPID value	Port Extender 802.1BR

Table 102: bcm_switch_control_set (Continued)

Global Control Type (bcm_switch_control_t)	Highlights	PG Document Reference Section
bcmSwitchL3UrpfDefaultRoute	Allow or forbid the Strict UC RPF packets to be forwarded by default route entry	Chapter 22, IP Router
bcmSwitchL3McRpfDefaultRoute	Allow or forbid the SIP-based MC RPF packets to be forwarded by default route entry	Chapter 22, IP Router
bcmSwitchMplsPWControlWord	Set Control-Value for Egress object MPLS-Port PWE	Chapter 26, VPLS and VPWS
bcmSwitchL2StationExtendedMode	Configure non-VRRP (l2_station) system mode	Chapter 22, IP Router
bcmSwitchArpMyIp1	Configure Ip1 of My-ARP protocol trap	Chapter 12, Traps
bcmSwitchArpMyIp2	Configure Ip2 of My-ARP protocol trap	Chapter 12, Traps
bcmSwitchArpIgnoreDa	Configure Ignore DA of My-ARP protocol trap	Chapter 12, Traps
bcmSwitchIcmpIgnoreDa	Configure Ignore DA of My-NDP protocol trap	Chapter 12, Traps
bcmSwitchL3McastL2	Configure default VSI forwarding type for IPMC-disabled IPv4 flow	Chapter 22, IP Router
bcmSwitchL3IP6McastL2	Configure default VSI forwarding type for IPMC-disabled IPv6 flow	Chapter 22, IP Router
bcmSwitchDescCommit	Commit the descriptor chain to HW and wait until it finishes	Chapter 6, Modular Database Profile
bcmSwitchL3RoutedLearn	Set routed learning mode for supported applications according to input flags	Chapter 21, Ethernet Bridge
bcmSwitchForwardLookupNotFoundTrap	Configure filtering of packets with a particular trap in case no valid forwarding destination is found. **NOTE:** IPv6 MC forwarding is not supported.	Chapter 22, IP Router
bcmSwitchFtmhEtherType	Set FTMH EtherType for remote CPU routing	Chapter 2, Ports and Generalized Ports
bcmSwitchIpv6NextProtocolEthernet	Configures the parser indication of next protocol Ethernet for IP tunnel, which can be either 59 or 143	Chapter 41, Segment Routing over IPv6
bcmSwitchSRV6EgressPSPEnable	Set Egress node PSP or USP mode	Chapter 41, Segment Routing over IPv6
bcmSwitchSRV6ReducedModeEnable	Set Ingress node: Normal or Reduced encapsulation mode	Chapter 41, Segment Routing over IPv6
bcmSwitchUdpTunnelIPv4DstPort	Configure UDP IPv4 destination port at parser to identify UDP next protocol	Chapter 29, IP Tunnel v4 Encapsulation
bcmSwitchUdpTunnelIPv6DstPort	Configure UDP IPv6 destination port at parser to identify UDP next protocol	Chapter 29, IP Tunnel v4 Encapsulation
bcmSwitchUdpTunnelMplsDstPort	Configure UDP Mpls destination port at parser to identify UDP next protocol	Chapter 29, IP Tunnel v4 Encapsulation
bcmSwitchVxlanUdpDestPortSet	Configure UDP VXLAN destination port at parser to identify UDP next protocol	Chapter 32, VXLAN IP Overlay
bcmSwitchVxlanGpeUdpDestPortSet	Configure UDP VXLAN GPE destination port at parser to identify UDP next protocol	Chapter 32, VXLAN IP Overlay
bcmSwitchHashIP4OuterField	Modify the bits of the outer IPv4 header which are used in the construction of the LB key	Chapter 13, Load Balancing
bcmSwitchHashIP4InnerField	Modify the bits of the inner IPv4 header which are used in the construction of the LB key	Chapter 13, Load Balancing

Table 102: bcm_switch_control_set (Continued)

Global Control Type (bcm_switch_control_t)	Highlights	PG Document Reference Section
bcmSwitchHashIP6OuterField	Modify the bits of the outer IPv6 header which are used in the construction of the LB key	Chapter 13, Load Balancing
bcmSwitchHashIP6InnerField	Modify the bits of the inner IPv6 header which are used in the construction of the LB key	Chapter 13, Load Balancing
bcmSwitchHashL2OuterField	Modify the bits of the outer L2 header which are used in the construction of the LB key	Chapter 13, Load Balancing
bcmSwitchHashL2InnerField	Modify the bits of the inner L2 header which are used in the construction of the LB key	Chapter 13, Load Balancing
bcmSwitchHashL4OuterField	Modify the bits of the outer L4 header which are used in the construction of the LB key	Chapter 13, Load Balancing
bcmSwitchHashL4InnerField	Modify the bits of the inner L4 header which are used in the construction of the LB key	bcmSwitchHashL4InnerField
bcmSwitchHashMPLSField0	Modify the bits of all MPLS labels which are used in the construction of the LB key	bcmSwitchHashL4InnerField
bcmSwitchBfdMyDipDestination	Set the value of the my-BFD-DIP destination	Chapter 34, BFD
bcmSwitchMplsAlternateMarkingSpecialLabel	Configure MPLS special label identification	Chapter 49, Instrumentation – Alternate Marking
bcmSwitchGlobalTodMode	Time of Day mode: ■ 0 – Null ■ 1 – IEEE 1588 (OAM) ■ 2 – NTP ■ 3 – Both	Chapter 33, Operations, Administration, and Maintenance
bcmSwitchControlOamBfdFailoverStateIgnore	Enable or disable ignoring protection state for OAMP injected packets	Chapter 33, Operations, Administration, and Maintenance
bcmSwitchIngParseL3L4IPv4	Disable IPv4oUDP parsing	—
bcmSwitchIngParseL3L4IPv6	Disable IPv6oUDP parsing	—
bcmSwitchL2LearnLimitCheckStatic	Enable or disable limit checking for static entry	Chapter 21, Ethernet Bridge
bcmSwitchL2ChangeFieldsEnable	Enable or disable update of existing static-transplantable L2 entries	Chapter 21, Ethernet Bridge
bcmSwitchIngressVlanEditClassNull	Enable or disable a reserved value to be a NULL VLAN edit class	Chapter 41, Segment Routing over IPv6
bcmSwitchTunnelRouteDisable	Enable or disable the tunnel termination process per specific layer above IPv4 or IPv6. The layer above is set with the BCM_SWITCH_TUNNEL_ROUTE_DISABLE_XXX flag.	Chapter 28, IP Tunnel v4 Termination
bcmSwitchMplsSpecialLabelPerPortTerminateDisable	Enable or disable per port control for special label auto termination.	Chapter 24, MPLS LER Termination