ZebOS-XP®
Network Platform

Version 1.4
Extended Performance

Carrier Ethernet
Developer Guide

December 2015

IP Infusion Inc.
3965 Freedom Circle, Suite 200
Santa Clara, CA 95054
+1 408-400-1900
http://www.ipinfusion.com/

For support, questions, or comments via E-mail, contact:
support@ipinfusion.com

Contents

Preface

This guide describes the ZebOS-XP application programming interface (API) for Carrier Ethernet.

Audience

This guide is intended for developers who write code to customize and extend Carrier Ethernet.

Conventions

Table P-1 shows the conventions used in this guide.

Table P-1: Conventions

Convention	Description
Italics	Emphasized terms; titles of books
Note:	Special instructions, suggestions, or warnings
`monospaced type`	Code elements such as commands, functions, parameters, files, and directories

Contents

This document contains these chapters and appendices:

- Appendix A, *G.8032 (ERPS) Version 1*

Related Documents

The following guides are related to this document:

- *Carrier Ethernet Command Reference*
- *Carrier Ethernet Configuration Guide*
- *Data Center Bridging Developer Guide*
- *Data Center Bridging Command Reference*
- *Data Center Bridging Configuration Guide*
- *Ethernet Local Management Interface Developer Guide*
- *Ethernet Local Management Interface Command Reference*
- *Ethernet Local Management Interface Configuration Guide*
- *Layer 2 Developer Guide*
- *Layer 2 Command Reference*
- *Layer 2 Configuration Guide*

Note: All ZebOS-XP technical manuals are available to licensed customers at http://www.ipinfusion.com/support/document_list.

Support

For support-related questions, contact support@ipinfusion.com.

Comments

If you have comments, or need to report a problem with the content, contact techpubs@ipinfusion.com.

CHAPTER 1 Introduction to Carrier Ethernet

Carrier Ethernet (CE) is a set of extensions and protocols required to enable service providers to support Ethernet services for customers and use Ethernet technologies in their networks.

Overview

The advantages of evolving Carrier Ethernet technologies are the varied services and applications that it can support, including:

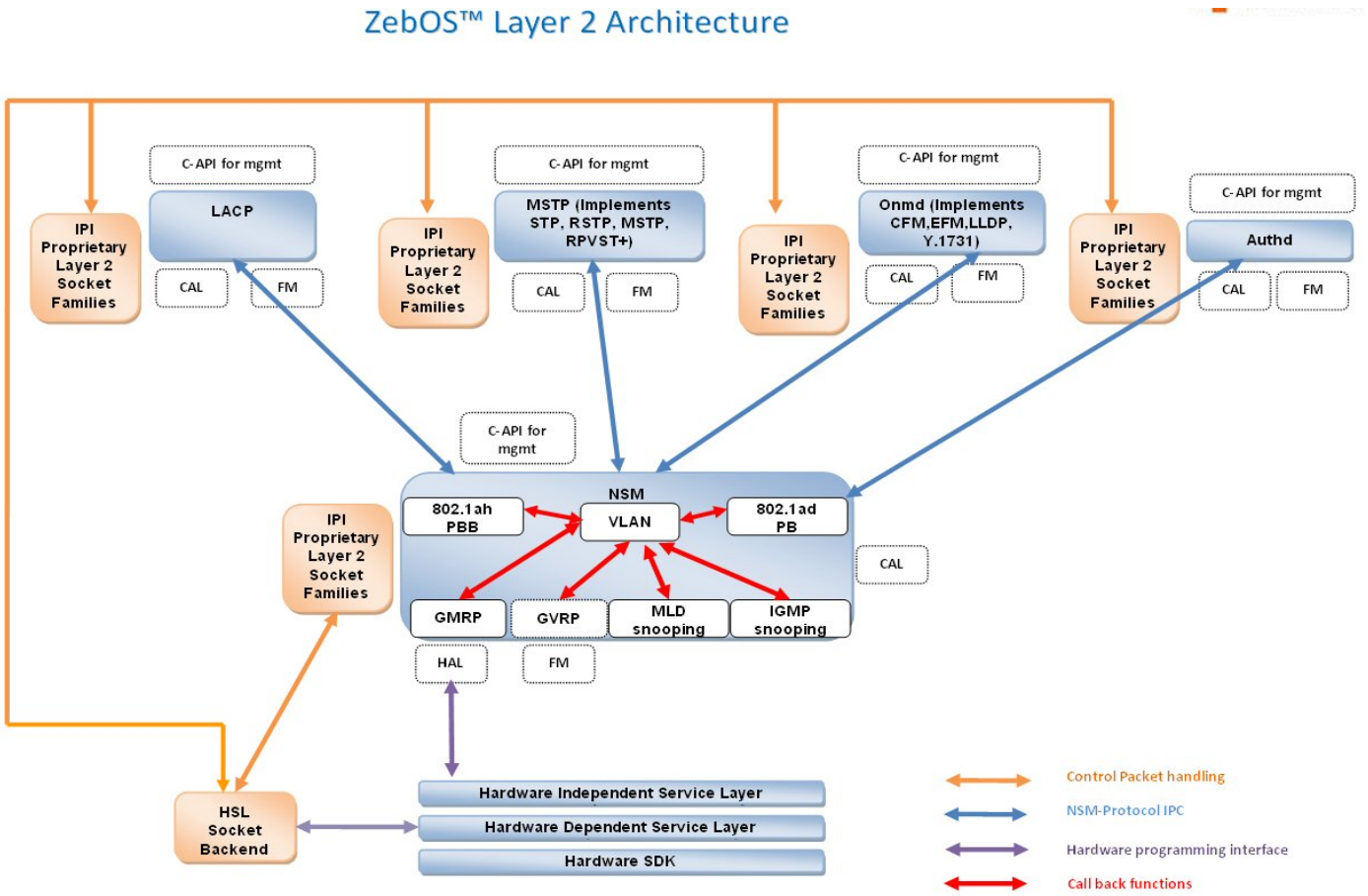
- LAN-to-LAN Networking
- Internet access
- IP Telephony (VoIP)
- Large file transferring and sharing
- Business security and disaster recovery
- Mission-critical application support across the enterprise or network
- Video conferencing and video security
- Storage NAS and SAN
- Distance learning

Customer Perspective

Providers and enterprises require the following abilities:

- Obtain bandwidth that is both flexible and scalable
- Improve network reliability
- Lower WAN and equipment costs
- Enable new business applications
- Reduce the complexity of module and equipment installations and upgrades, without negatively impacting clients
- Provide quality of service (QoS)

Architecture



Supported Modules

ZebOS-XP Carrier Ethernet provides the modules as described in the chapters below:

Chapter	Contents
Chapter 2, *Connectivity Fault Management*	Connectivity Fault Management (CFM) that provides proactive connectivity monitoring, fault verification, and fault isolation
Chapter 3, *Provider Bridging*	Provider Bridging which enables providers to supply Ethernet Virtual Circuit (EVC) services to their customers
Chapter 4, *Provider Backbone Bridging*	Provider Backbone Bridging which enables a provider to build an Ethernet backbone network of multiple, inter-connecting Provider Bridging networks
Chapter 5, *Provider Backbone Bridge - Traffic Engineering*	Provider Backbone Bridge - Traffic Engineering (PBB-TE) features for CFM protocols and addressing of PBB-TE MEPs, MIPs, and TE Service Instances
Chapter 8, *Ethernet Protection Switching*	Point-to-point Ethernet path protection
Chapter 9, *G.8032 (ERPS) Version 2*	Ethernet rings
Chapter 10, *Ethernet to the First Mile*	Ethernet to the First Mile Operations, Administration, and Management

Chapter	Contents
Chapter 11, *Link Layer Discovery Protocol*	How bridges on a LAN send and receive connectivity and management related information to each other
Chapter 13, *User Network Interface*	The demarcation point between service providers and subscribers

Supported Standards

For information on the supported standards for Carrier Ethernet, see the *Architecture Guide* and *Feature Matrix*.

Common Data Structures

This section describes data structures that are referenced in more than one chapter of this guide.

See the *Common Data Structures Developer Guide* for a description of these data structures used by multiple ZebOS-XP modules:

- `cli`
- `interface`
- `lib_globals`
- `nsm_bridge`
- `nsm_bridge_master`
- `variable`

onmd_bridge

This data structure in `onmd/onmd.h` holds information about an onmd bridge.

Member	Description
`name`	Bridge name
`bridge_type`	Whether this is a PBB bridge and if so whether it is an i-comp or b-comp bridge
`is_edge`	Whether this is a Provider Edge Bridge
`port_list`	Port list
`vlan_table`	VLAN tree indexed on vid
`cfm`	Pointer to CFM list
`te_sid_list`	This list has onm_pbb_te_sid structure nodes which are created as soon as a TE-SID is configured at NSM
`esp_vlan_tree`	Ethernet Switched Path VLAN tree
`eps_tree`	Ethernet Protection Switching tree

Member	Description
raps_tree	Pointer to g8032 logical rings
te_aps_tree	TE Automatic Protection Switching tree

```
struct onmd_bridge
{
  /* Bridge Name */
  u_char name[L2_BRIDGE_NAME_LEN + 1];
  /* To identify whether it is a PBB bridge and if so, it is
   *  i-comp or b-comp bridge */
  u_int8_t bridge_type;
  /* Whether this is a Provider Edge Bridge */
   u_int8_t is_edge;
  /* Port list */
  struct list *port_list;
  /* VLAN tree indexed on vid. */
  struct route_table *vlan_table;
#ifdef HAVE_CFM
  /* Pointer to CFM list */
  struct cfm *cfm;
#ifdef HAVE_PBB_TE
  /* ETH_0040_PBB_TE_MIP Start*/
  /* This list has the onm_pbb_te_sid structure nodes, which will be created
   * as soon as a TE-SID is configured at NSM */
  struct avl_tree *te_sid_list;
 /* ETH_0040_PBB_TE_MIP End */
#if defined HAVE_I_BEB && defined HAVE_B_BEB
  u_int8_t *esp_vlan_tree [ONM_VLAN_MAX];
#endif /* HAVE_I_BEB && HAVE_B_BEB */
#endif /* HAVE_PBB_TE */
#endif /* HAVE_CFM */
#ifdef HAVE_G8031
  struct avl_tree * eps_tree;
#endif /* HAVE_G8031 */
#ifdef HAVE_G8032
  /* pointer to g8032 logical rings defined */
    struct avl_tree *raps_tree;
#endif /*HAVE_G8032*/
#if defined HAVE_CFM && defined HAVE_I_BEB && defined HAVE_I_BEB && \
    defined HAVE_PBB_TE
  struct avl_tree * te_aps_tree;
#endif /* HAVE_CFM && HAVE_I_BEB && HAVE_B_BEB && HAVE_PBB_TE */
};
```

CHAPTER 2 Connectivity Fault Management

Ethernet Connectivity Fault Management (CFM) is an end-to-end, per-service instance Ethernet layer Operations and Management (OAM) protocol that includes proactive connectivity monitoring, fault verification, and fault isolation.

Overview

Ethernet CFM provides the following benefits:

- End-to-end service-level OAM technology
- Reduced operating expense for service provider Ethernet networks
- Competitive advantages for service providers
- Enables CFM services in Provider Bridges (PB), Provider Backbone Bridges (PBB) and PBB-TE

A customer service instance, or Ethernet Virtual Connection (EVC), as defined by the Metro Ethernet Forum (MEF), is a service that is sold to a customer. It is identified by a by an S-VLAN (service VLAN) tag on the User-to-Network Interface (UNI). EVC is natively switched via Ethernet or mapped to a pseudo-wire. The CFM protocols help detect, verify, and isolate connectivity failures in bridged VLANs, provider bridges, and provider backbone bridges. Functions can be managed by multiple independent organizations, each with restricted access to each other's equipment.

The ZebOS-XP CFM implementation fully complies with these standards:

- IEEE P802.1ag-2007, *Standard for Local and Metropolitan Area Networks — Virtual Bridged Local Area Networks — Amendment 5: Connectivity Fault Management*. The standard defines entities, protocols, procedures, and managed objects to support CFM, allow discovery and verification of frame paths to and from specific network users, and detect and isolate of connectivity faults to a specific bridge or LAN.
- ITU-T *Recommendation Y.1731, OAM Functions and Mechanisms for Ethernet-based Networks*. The Recommendation describes support for mechanisms required to operate and maintain the network and service aspects of the ETH layer. The ITU-T Recommendation complements the *IEEE 802.1ag Connectivity Fault Management* specification.

Features

The features for CFM include the following:

- Multiple VLAN IDs (VID) per Maintenance Association (MA)
- Link-Level Maintenance End Points (MEP)
- Default Maintenance Domain (MD) Level
- Maintenance Domain and Maintenance Association (MA) Name Types
- Maintenance Intermediate Point (MIP) creation permissions
- MEP Attributes
- Provider Backbone Bridge-specific (PBB) CFM (for PBB-specific features, please refer to Chapter 4, *Provider Backbone Bridging*)
- CFM MIB
- Support for both distribution and access network environments with enhancement of outward-facing Maintenance End Points (MEP)

CFM Managed Objects

The following lists the relationships for the managed objects for bridges:

- Maintenance Domain List — one list per bridge that contains the list of domains configured on the bridge
- CFM Stack — one set per bridge
- Default MD Level — one set per bridge
- Configuration Error List — one set per bridge

Maintenance Domains contain the following managed objects:

- Maintenance Associations — one set per MD
- Maintenance association End Points — one set per MA

CFM Entities

Major CFM entities are defined in the sections that follow.

Maintenance Domain

A maintenance domain (MD) is a part of the network that is controlled by a single operator. An MD is used to support connectivity between the Domain Service Access Points (DSAP) that is bound to that domain.

An MD is defined by the set of ports internal to it and at its boundary. A unique maintenance level in the range of 0 to 7 is assigned to each MD by a network administrator. Levels and domain names are useful for defining the hierarchical relationship that exists among domains. A customer domain may have a maintenance level of 7 and an operator domain may have a maintenance level of 0.

Maintenance Domain Level

MD Level information is carried in CFM PDUs. It allows each operator’s customers to use CFM functions. Customers may also behave as operators when necessary.

Maintenance Association

A Maintenance Association (MA) is a set of MEPs, each configured with the same MA Identifier and MD Level. A MA is used to monitor connectivity provided by that instance throughout the maintenance domain.

Maintenance Association Identifier

A Maintenance Association Identifier (MA ID or MAID) is a unique identifier within a domain that CFM uses to protect against the accidental concatenation of service instances. At the innermost MD level, every bridge port is enabled to provide IP Service on Any Port (ISAP) and can be configured as a Maintenance Intermediate Point (MIP). A MIP helps detect connectivity failures between any pair of MEPs in an MA, and isolates the failure to smaller segments. MEPs form the end points in an MA, whereas MIPs are the intermediate points along the path between MEPs. An MIP can also be configured as a MEP for its lower domain. There can be any number of MEPs and MIPs within an MA.

Customer Service Instance

A customer service instance is an Ethernet virtual connection (EVC), which is identified as an S-VLAN (service VLAN) within an Ethernet island, and is identified by a globally unique service ID.

Maintenance Points

A maintenance point (MP) is a demarcation point on an interface that participates in CFM within a maintenance domain. Maintenance points on device ports act as filters that confine CFM frames within the bounds of a domain by dropping frames that do not belong to the correct level.

An MP consists of a shim with two Service Access Points (SAP) to pass frames between it and other entities.

MEPs have an inner and an outer SAP, whereas an MIP has two components called MIP Half Functions (MHF) that operate in a manner similar to a maintenance point. There are two types of maintenance points

- Maintenance End Points (MEP)
- Maintenance Intermediate Points (MIP)

Maintenance End Point

a MEP is identified per-maintenance domain and service provider VLAN (S-VLAN). It is present at the edge of a domain, and defines the boundary of the domain. Its functionality includes confining the messages for a particular level to within the domain of that level. However, higher level messages can pass through lower level domains transparently.

MEPs form the end points in an MA, whereas MIPs are intermediate points in the path between MEPs. An MIP can also be configured as a MEP within its lower MD. Any number of MEPs and MIPs are allowed within a maintenance association. All MEPs are identified by unique MEP IDs.

A MEP with its outer SAP closer to the MAC relay entity than its inner SAP is called a “Down MEP”. When the outer SAP of the MEP is farther away from the MAC relay entity than the inner SAP, it is called an “Up MEP”.

A maintenance end point has either an Up or Down MHF entity. These entities only pass, direct or filter frames without altering them.

Inward-Facing UP MEP

An inward-facing, or UP MEP, communicates through the bridge relay function, thus a MEP resides in a Bridge that transmits CFM Messages towards, and receives them from, the direction of the Bridge Relay Entity.

Outward-Facing DOWN MEP

An outward-facing, or DOWN MEP, communicates through the wire, and is a MEP that resides in a Bridge that transmits CFM Messages towards, and receives them from, the direction of the physical medium.

Maintenance Intermediate Point

An MIP is valid per maintenance domain (level), and is available for all S-VLANs enabled or allowed on a port. It is internal to a domain and is not present at the boundary. All CFM frames at a lower level are stopped and dropped, independent of whether they originate from a wire or relay function. All CFM frames at a higher level are forwarded. MIPs are passive points, which means that they respond only when triggered by a CFM Traceroute or Loopback message (TRM or LBM). An MIP consists of one or more MIP Half Functions.

An MIP can be set to be created automatically or to be configured manually. The ZebOS-XP Layer 2 Forwarder untagged CFM frames to be sent if the VID passed is 0, regardless of whether the port has been configured as a trunk or access port.

MIP Half Function

An MIP Half Function (MHF) is a CFM entity associated with a single MD, thus with a single MD Level and a set of VIDs. An MHF can generate CFM Messages, but only in response to received CFM Messages.

Maintenance Entity

A maintenance entity (ME) is a point-to-point relationship between two MIPs within a single maintenance association.

CFM Processes

Connectivity Fault Management uses the following processes to identify, verify and submit notifications about the faults that have occurred in the bridged LAN in which it is configured.

Connectivity Check Messages

Continuity Check Messages (CCM) are multicast heartbeat messages that are exchanged periodically among MEPs. CCMs allow:

- MEPs to discover other MEPs within a domain
- MIPs to discover MEPs.
- MEPs to detect loss of service connectivity amongst themselves.

CCMs are confined to a domain and S-VLAN. They allow full-transmission interval support and point-to-point unicast support.

Each MEP is uniquely identified by an ID that is associated to a MAID and an MD level. The MEPIDs for all MEPs in an MA are configured on every other MEP. Every MEP generates a single CCM, and sends it to a multicast destination MAC address.

Upon receipt of a CC message, the MEP checks with its own list of MEPIDs, and performs other validation functions to detect any fault. When three consecutive CCMs from a targeted MEP are not received, a connectivity error is detected for that MEP.

There are two types of traps:

- Continuity-check trapping is initiated when a new MEP is discovered, a MEP is down, or when cross-connect is detected.
- Cross-check trapping is initiated when a service is enabled, a remote MEP is missing, or an unknown MEP is discovered.

Loopback Messaging

Loopback messages (LBM) are transmitted by a MEP at the request of the administrator to verify connectivity to a particular MIP or MEP. An LBM indicates whether the target maintenance point is reachable or not. It does not allow hop-by-hop discovery of the path, and is similar in concept to ICMP Echo (ping).

A MEP can optionally use a Data TLV or Test TLV.

- When configured for validating the successful transmission of different frame sizes, the MEP uses a Data TLV.
- When configured for diagnostic tests, the MEP transmits a Unicast Loopback Message (LBM) frame addressed to the remote peer MEP with a Test TLV.

The Multicast LB function is used to verify bidirectional connectivity of a MEP with its peer MEPs. Multicast LB is an on-demand OAM function. When a Multicast LB message is invoked by a MEP, the MEP returns to the initiator of the Multicast LB message a list of peer MEPs with whom bidirectional connectivity is detected.

When an LBM is received by a MEP or MIP, it sends a loopback response (LBR).

Link Trace Messaging

Linktrace messages (LTM) are transmitted by a MEP at the request of the administrator to track the path (hop-by-hop) to a destination MEP. They allow the transmitting node to discover vital connectivity data about the path. The concept is similar to UDP Traceroute.

An LTM is generated by setting the target MAC address and sending it to a multicast destination address. The MIPs in the path to the destination MEP reply to the originating MEP with a unicast Linktrace Reply (LTR) message.

Fault Alarm Notification

Fault alarm notifications are transmitted by a MEP. If a fault is detected by a MEP and the priority of the highest defect is greater than the lowest priority for which it is qualified to be notified, then a fault alarm is generated for the highest defect.

Frame Loss Measurement

Ethernet OAM Frame Loss Measurement (LMM) is used to collect counter values applicable to ingress and egress service frames where the counters track the number of transmitted and received data frames between a pair of MEPs. A MEP sends frames with LM request information to its peer MEP, and receives frames with LM Reply (LMR) information from the peer MEP to calculate loss measurements.

Frame Delay Measurement

Frame delay and frame delay variation measurements are performed by sending periodic frames via an Ethernet Delay Message (DM) to the peer MEP and receiving frames with Ethernet DM information from the peer MEP during a configured diagnostic interval. Frame delay and frame delay variation can be performed in two ways, one-way DM (1DM) or two-way DM (DMM):

- For 1DM, a MEP sends frames with one-way DM data to its peer MEP in a point-to-point ME to measure one-way frame delay and/or one-way frame delay variation measurements at the peer MEP.
- For DMM, a MEP sends frames with DM request information to its peer MEP and receives frames with DM reply (DMR) information from the peer (DVM).

Lock Signal

Ethernet Lock Signal (LCK) is used to communicate the administrative locking of a server sub-layer MEP, and consequential interruption of data traffic forwarding to the MEP that is expecting traffic.

When LCK is enabled, LCK frames are sent to the given destination address. Upon receiving the LCK signal, a message is logged.

For an UP MEP LCK transmitter, LCK is generated towards the interface (port) on which the UP MEP is configured on. For a DN MEP LCK transmitter, LCK is generated towards the MAC relay.

Test Signal

Ethernet Test Signal (TST) is used to perform one-way, on-demand, in-service or out-of-service diagnostic tests. When the out-of-service TST function is performed, client data traffic is disrupted in the diagnosed entity. a MEP configured for the out-of-service test transmits LCK frames. A TST frame can be sent once to the specified destination, or if the recursive option is used, then frames are sent at periodic intervals.

Alarm Indicator Signal

Ethernet Alarm Indication Signal (AIS) is used to suppress alarms following detection of defect conditions at the server sub-layer. Transmission of frames with AIS information can be enabled or disabled on a MEP. The expected result is that lower level alarms should not trigger higher level MEPs and MIPs to generate alarms. The rationale is that an alarm in the providers' network should not alter customer equipment, so it is easier to identify whose (provider or customer) network is causing the fault.

Once AIS is enabled for a server MEP on an interface, AIS frame generation is triggered by an Ethernet interface link down event. When AIS is enabled on a MEP, when a CCM LOS is detected or an AIS or LCK frames are received, AIS frames are generated.

AIS transmission terminates when the defect condition is removed, or AIS generation is disabled using the CLI.

Maintenance Communication Channel

Ethernet Maintenance Communication Channel (MCC) is used to perform remote management between a pair of MEPs. The ZebOS-XP MCC implementation handles just the messaging function, which means that an MCC from a remote MEP is identified as an MCC frame, but no payload handling is done.

Experimental OAM

Experimental OAM (EXM) functionality can be used within an administrative domain on a temporary basis. The ZebOS-XP EXM implementation handles the message part of the frame, which means that it is capable of identifying an EXM frame, but does not handle the payload.

Vendor-Specific OAM

Vendor-specific OAM (VSM) functionality may be used by a vendor across its equipment. Interoperability of vendor-specific OAM functionality is not expected across different vendors' equipment. The ZebOS-XP VSM implementation handles the message part of the frame, which means that it is capable of identifying a VSM frame, but does not handle the payload.

Throughput Measurement

Throughput is defined as the maximum rate at which no frame is dropped. Throughput measurement is done by sending frames at increasing rates (up to a theoretical maximum), graphing the percentage of frames received at each rate, and reporting the rate at which frames start being dropped. In general, the rate is dependent on the frame size.

Multiple VIDs Per Maintenance Association

The concept of monitoring multiple VIDs per Maintenance Association follows this logic:

- An MA is uniquely identified in an MD and assigned to be the Primary VID. Multiple secondary VIDs can be attached to the MA.
- A MEP in the MA inherits all the VIDs identified, both primary and secondary. a MEP can either use the primary VID of the MA, or a primary VID can be locally configured for the MEP. a MEP can *send* CFM frames only via the primary VID (PVID). However, a MEP can *receive* frames from all secondary VIDs.

The advantage is that only one MEP needs to be configured to monitor *n*-VIDs on a port. The following diagram represents MA 12 with three bridges, A, B, and C. Three Up MEPs are configured in MA 12 (identified by the yellow triangles), one per bridge.

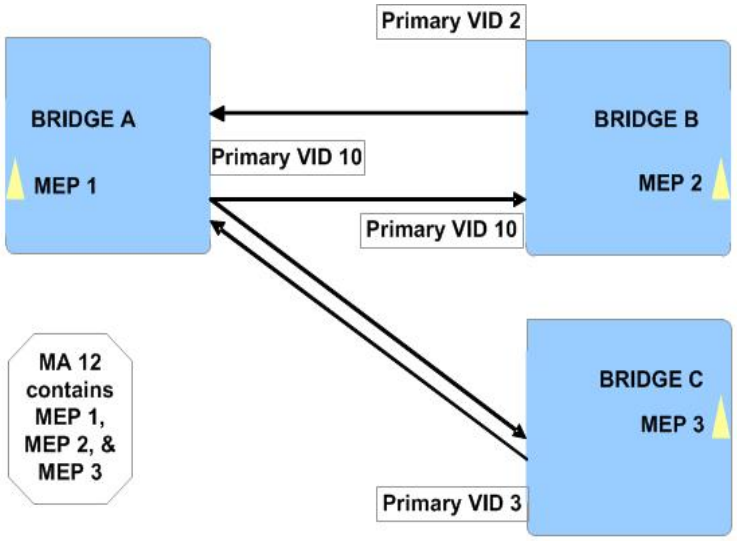


Figure 2-1: Multiple VID in a Single Maintenance Association

Note: In this diagram, MEP 1 acts as the starting point to illustrate the functions in MA 12. MEP 2 and MEP 3 are Remote MEPs (RMEP) associated to MEP 1 within MA 12.

In this configuration:

- MA 12 is configured with the Primary VID 10.
- On bridge A, Up MEP 1 is configured, and it inherits the primary VID from MA 12.
- Primary VID 2 and Primary VID 3 are secondary VIDs for MEP 1.
- MEP 1 can send CFM frames via VID 10 to the remote Up MEPs 2 and 3, but can only receive frames from VID 2 and VID 3.
- On Bridge B and Bridge C, MEP 2 and MEP 3 are configured as local Primary VID 2 and Primary VID 3, respectively.
- Although MA 12's primary VID on Bridge B and Bridge C is 10, the MEPs on Bridge B and Bridge C can be configured to send frames via VID 2 and VID 3, which are secondary VIDs within MA 12.

This design permits Primary VID and Secondary VID mapping is as follows:

- When a VLAN is configured on a bridge, its primary VID is the same as its VLAN identifier. When a VID is configured as a secondary VID, its primary VID is the VLAN identifier of the primary VLAN.
- An MA can be configured only on primary VIDs.
- When a secondary VID is deleted from a VLAN, its primary VID is returned to its own `vlan_id`.

Link-Level Maintenance End Point

A link-level MEP does not monitor VLANs. Instead, it monitors the health of the physical link. In the figure, the link-level MEP is the Down MEP in position 5.

Note: Down MEPs are indicated by the shaded inverted triangles.

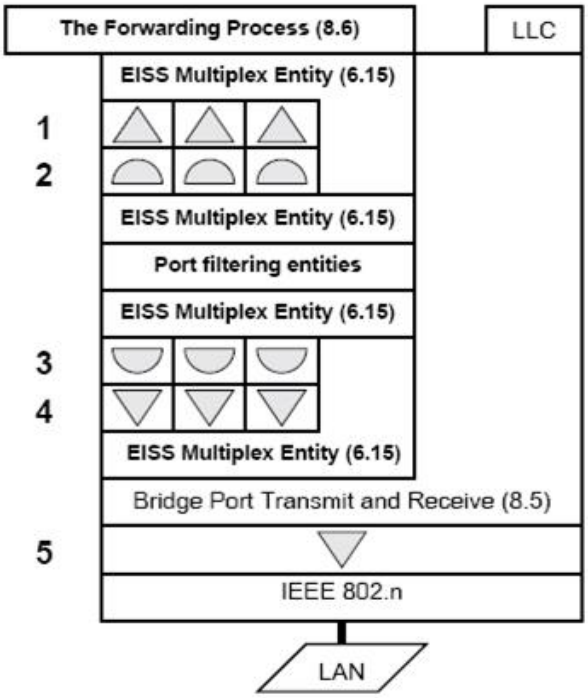


Figure 2-2: Illustration of Link-Level MEP

In order to configure a Down MEP in position 5, an MA needs to be created.The identifier for a link-level MEP is not the MD level 0, but an MA for which no VID has been configured.

A link-level MEP sends out only untagged frames. The ZebOS-XP Layer 2 Forwarder allows untagged CFM frames to be sent when the VLAN identifier is 0, regardless of whether the port has been configured as a trunk or an access port.

Upon receiving an untagged frame, the L2 forwarder adds the PVID of the port, and passes the frame to the CFM module. In `onmd`, the frame is received as a tagged frame. When `onmd` receives an untagged frame, a bit is set so that when the frame is received by CFM, the protocol recognizes it as an untagged link-level frame.

Default MD Level

The purpose of the Default MD Level entity is to control MIP creation on a bridge of VIDs that do not belong to any MA. There is a single Default MD Level managed object per bridge. A Default MD Level table is maintained for each bridge. When initialized, a process creates the table automatically with entries for all VLAN IDs and default values specified for each object. An entry in the Default MD Level table is configured, and MIP creation permissions are set. Changes can only be made by the operator. An entry for a VID is created when the VID is added to a bridge, and entries in the Default MD Level table are indexed by VID.

MIP Creation

MIP creation is indirectly controlled by the Default MD Level object. The following conditions trigger the management entity to evaluate the creation of MIPs for a VID on a bridge port:

- A Bridge is initialized
- a MEP is created on or deleted from VID(s) *x* on a Bridge Port
- An Up MEP is created on or deleted from VID(s) *x on any bridge port*
- A change occurs in the state for which VID(s) *x* can or cannot pass frames through the Bridge Port (applies to MAs not associated with any VID)
- A change occurs in the default MD level managed object
- A change occurs in an MA managed object associated with VID *x*.

Active Levels List

When required, the Maintenance Entity performs an MIP evaluation on each Bridge Port for each VID. For each Bridge Port *p* and VID *x*, the Maintenance Entity creates a list of active MD Levels. The list contains:

- the MD Level of each of the MAs that includes VID *x* and a MEP configured on Bridge Port *p*
- the MD Level of each of the MAs that includes VID *x* and has at least one Up MEP configured on any Bridge Port in this Bridge other than Port *p*
- the MD Level of each of the MAs that include VID *x*, and has no MEPs configured on any Bridge Port in this Bridge
- the MD Level of the entry in the Default MD Level managed object for VID *x*.

Exactly one MD Level, identified as *d* in the discussion, in the Active Levels List, is eligible for MIP creation. It is the lowest MD Level in the list of Active MD Levels, so no MEP is configured on Bridge Port *p* or VID x at MD Level *d,* or at any higher MD Level. This is achieved as follows:

- An Active level list is maintained on each port.
- On each port, there is a list in which each node consists of a VID and the Active Level for that port.

With this information, the Active levels for the VID on that port can be determined.

MIP Creation Decision Matrix

An MIP is created for each VID on a port. The active-level computation is VID- and Port-based, so, on a port for a VID there can only be one VID. If the VID is configured as a Primary VID for an MA, then the MIP is also responsible for the entire set of secondary VIDs mapped to the primary VID.

The following list defines each of the MIP creation permission values.

- `CFM_DEF_MHF_NONE`: No MIP Half Functions (MHF) can be created for the VID(s) (the default value).
- `CFM_DEF_MHF_DEFAULT`: MHFs can be created for the VID(s) on any Bridge Port through which the VID(s) can pass where:
  - there are no lower active MD levels; or
  - there is a MEP at the next lower active MD level on the port.
- `CFM_DEF_MHF_EXPLICIT`: MHFs can be created for the VID(s) only on Bridge Ports through which the VID(s) can pass, and only if there is a MEP at the next lower Active MD level on the port.
- `CFM_DEF_MHF_DEFER`: In the Maintenance Association managed object only, the control of MHF creation is deferred to the corresponding variable in the enclosing Maintenance Domain.

The table below lists the conditions under which MIPs are created.

Enumeration	MEP is Configured on Bridge Port *p*	MIP Created
CFM_DEF_MHF_NONE	—	No
CFM_DEF_MHF_EXPLICIT	False True	No Yes
CFM_DEF_MHF_DEFAULT	—	Yes

Traceroute Implementation

Traceroute implementation for CFM emulates that of the `show` commands used in ZebOS-XP. While parsing a command from the command line interface (CLI), if a sequence is identified, then the command is handled like the `show_read` command. In turn, the `show_out` command is called, which is handled in such a way that a traceroute node is passed as an `arg`. When a response is received, it is displayed and the command returns a success message.

When the CLI status is `CLI_CONTINUE`, the `write` thread is called so that control is returned only after a response is displayed on the console. Ping and Traceroute are implemented as separate commands.

Data Structures

The functions in this chapter refer to the data structures described in this section.

cfm_1dm_rx

This structure in `onmd/cfm/cfm_dm.h` defines a one way delay measurement (1DM) receive frame.

Member	Description
mep	Back pointer to the MEP
dm_rx_while	Thread for receiving 1DM frames for a given duration
dm_rmep_list	Linked list in which RMEP information from which 1DM frames are received is dynamically kept
dm_active	For CLI blocking

```
struct cfm_1dm_rx
{
  /* Back pointer to the MEP */
  struct cfm_mep *mep;
  /* Thread for receiving 1DM frames for a given duration */
  struct thread *dm_rx_while;
  /* This is the linked list in which rmep information from which 1DM frames are
received will be dynamically kept */
  struct list *dm_rmep_list;
```

```
  /* For CLI blocking */
  u_int8_t dm_active;
};
```

cfm_dmm

This structure in `onmd/cfm/cfm_dmm.h` defines a two way delay measurement (dmm) frame.

Member	Description
mep	Back pointer to the MEP
dmm_while	Thread for sending frames periodically
dest_mac	Destination MAC address
dmm_frames_to_be_sent	Total number of frames sent based on the time value for which DMM was enabled
dmm_data	Frame data to be sent
dmm_interval	Transmission interval
dmm_type	Unicast or multicast frame
dmm_active	For CLI blocking
cli	For CLI blocking
dmm_indx	Index into the instance of the last DMM frame sent
cfm_dmm_frame_instance	Pointer to array of dynamically allocated instances for struct cfm_dmm_frame

```
struct cfm_dmm
{
  /* Back pointer to the MEP */
  struct cfm_mep *mep;
  /* Thread for sending frames periodically */
  struct thread *dmm_while;
  /* Destination Mac address */
  u_char dest_mac[MACADDR_LEN];
  /* Number of frames DMM frames to be sent based on the time duration for which two-way
DM is enabled */
  u_int16_t dmm_frames_to_be_sent;
  /* Pointer in which frame to be filled */
  u_char *dmm_data;
  /* Transmission Interval */
  u_int16_t dmm_interval;
  /* unicast or multicast frame */
  u_int8_t dmm_type;
  /* For CLI blocking */
  u_int8_t dmm_active;
  struct cli *cli;
  /* Index into the instance of the last DMM frame sent */
```

```
  u_int8_t dmm_indx;
  /* Pointer to array of dynamically allocated instances for struct cfm_dmm_frame */
  struct cfm_dmm_frame *cfm_dmm_frame_instance[CFM_DMM_MAX_FRAME_INSTANCES];
};
```

cfm_lb

This structure in `onmd/cfm/cfm_lb.h` defines a CFM loopback.

Member	Description
`mep`	Pointer to MEP
`lb_len`	Loopback length
`lb_data`	Loopback data
`lbms_to_send`	Loopback message to send
`next_lbm_transid`	Next loopback message transaction identifier
`expected_lbr_transid`	Expected loopback receiver transaction identifier
`tx_lbm_messages`	Number of messages to send
`lbi_active`	Active loopback identifier
`lb_while`	Thread for maintaining timers to send out frames periodically
`lb_response`	Loopback response
`xmit_ready`	Transmit ready
`remac`	MAC address
`lbm_tx_int`	Loopback message interval
`lbr_timeout_int`	Loopback timeout interval
`in_order_lbr`	In order loopback reply
`curr_in_order_lbr`	Current in order loopback reply
`out_order_lbr`	Out order loopback reply
`mismatch_lbr`	Mismatched loopback reply
`lbr_sent`	Loopback reply sent
`curr_lb_lbr_recv`	Current loopback reply received
`lbr_bad_msdu`	Number of loopback responses received with corrupted data; need hardware integration to use this
`tx_lbm_result_ok`	True if MEP is active, otherwise false
`tlv_type`	ITU-T Y1731 section 7.2.1.1: whether the TLV type is 3 (data) or 32 (TST)

Member	Description
tlv_length	The length of TLV
tlv_tst_pattern	Test TLV pattern
lb_type	ITU-T Y1731 section 7.2.2: whether the LBM is unicast or multicast
rmep_lbr_count	Number of remote MEPs that have replied to the multicast LBM
lb_rmep_list	Linked list to have information about RMEPs sending LBRs for multicast LBM
recursive_mcast	Whether multiple multicast frames were sent
lbm_pbb_te_mip_tlv	IEEE-802.1Qay-d3-5 clause 21.7.5: PBB-TE MIP TLV

```
struct cfm_lb
{
  /* Pointer to MEP */
  struct cfm_mep *mep;
  u_int32_t lb_len;
  char *lb_data;
  u_int32_t lbms_to_send;
  u_int32_t next_lbm_transid;
  u_int32_t expected_lbr_transid;
  /* number of messages configured to be sent */
  u_int16_t tx_lbm_messages;
  u_char lbi_active;
  struct thread *lb_while;
  struct thread *lb_response;
  u_int8_t xmit_ready;
  u_int8_t remac[MACADDR_LEN];
  u_int8_t lbm_tx_int;
  u_int8_t lbr_timeout_int;
  u_int32_t in_order_lbr;
  u_int32_t curr_in_order_lbr;
  u_int32_t out_order_lbr;
  u_int32_t mismatch_lbr;
  u_int32_t lbr_sent;
  u_int8_t  curr_lb_lbr_recv;
  /* For now this value is 0, need hardware integration to use this */
  s_int32_t lbr_bad_msdu;
  /* As of now this is true if mep is active else false */
  bool_t tx_lbm_result_ok;
  /* For ITU-T Y1731 section 7.2.1.1 */
  /* To identify whether the TLV Type is 3 (data) or 32 (TST) */
  u_int8_t tlv_type;
  /* The length of TLV */
  u_int16_t tlv_length;
  /* Test TLV Pattern */
  u_int8_t tlv_tst_pattern [CFM_TEST_TLV_PATTERN_SIZE];
#if defined HAVE_CFM_Y1731 || defined HAVE_PBB_TE
```

```
  /* For ITU-T Y1731 section 7.2.2 */
  /* To distinguish the LBM as unicast or multicast */
  u_int8_t lb_type;
  /* Number of remote MEPs that have replied to the multicast LBM */
  u_int16_t rmep_lbr_count;
  /* Linked list to have information about RMEPs sending LBRs for mcast LBM */
  struct list *lb_rmep_list;
  /* To know whether multiple mcast frames were sent */
  u_int8_t recursive_mcast;
#endif /* HAVE_CFM_Y1731 || HAVE_PBB_TE*/
#if defined HAVE_CFM && defined HAVE_I_BEB && defined HAVE_B_BEB && \
    defined HAVE_PBB_TE
  /* PBB-TE MIP TLV as per the clause 21.7.5 in IEEE-802.1Qay-d3-5 */
  struct cfm_pbb_te_mip_tlv lbm_pbb_te_mip_tlv;
#endif /*HAVE_CFM && defined HAVE_I_BEB && defined HAVE_B_BEB && HAVE_PBB_TE */
};
```

cfm_lm

This structure in `onmd/cfm/cfm_lm.h` defines a CFM frame loss measurement.

Member	Description
`mep`	Back pointer to MEP that owns this structure
`lm_while`	Thread for maintaining timers to send out frames periodically.
`lm_data`	Pointer in which the frame is filled to be sent
`destAddr`	Destination MAC address
`tx_interval`	Transmission interval for frames (default is one second)
`lm_type`	Whether unicast or multicast frames are sent
`lm_active`	For CLI blocking
`lm_frames_to_be_sent`	Total number of frames to send based on the time duration for which LM was enabled
`lm_indx`	Index into the instance of the last LMM frame sent
`lm_matched_indx`	Index of the cfm_lm_counter instance for which LMR has come in
`sid`	For PBB
`cfm_lm_counter_instance`	Pointer to array of dynamically allocated instances for struct cfm_lm_counter

```
struct cfm_lm
{
  /* Back pointer to MEP */
  struct cfm_mep *mep;
  /* Timer thread for periodically sending out LMM */
  struct thread *lm_while;
```

```
  /* Pointer to form the frame and send */
  u_int8_t *lm_data;
  /* Destination MAC address */
  u_int8_t destAddr[MACADDR_LEN];
  /* Transmission interval for LMM frames */
  u_int16_t tx_interval;
  /* Unicast or Multicast LMM frames to send */
  u_int8_t lm_type;
  /* For CLI blocking */
  u_int8_t lm_active;
  /* Total number of frames to be sent based on the time duration for which LM has been
enable */
  u_int8_t lm_frames_to_be_sent;
  /* Index into the instance of the last LMM frame sent */
  u_int8_t lm_indx;
  /* Index of the cfm_lm_counter instance for which LMR has come in */
  u_int8_t lm_matched_indx;
  /* For PBB this field will be used */
  u_int16_t sid;
  /* Pointer to array of dynamically allocated instances for struct cfm_lm_counter */
  struct cfm_lm_counter *cfm_lm_counter_instance[CFM_LM_MAX_COUNTER_INSTANCES];
};
```

cfm_md

This structure in `onmd/cfm/cfmd.h` defines a CFM maintenance domain (MD).

Member	Description
`name`	MD Name
`name_type`	MD name type
`mac_num`	MAC domain name type only: two octet integer
`md_name_mac`	MAC domain name type only: MAC address
`level`	MD level
`bridge`	Bridge*
`ma_name_tree`	Pointer to MA list: accessed by vid
`ma_vid_tree`	Pointer to MA list: accessed by name
`ma_index_bmp`	Indexes for all the MAs; used for SNMP
`md_index`	MD index
`pbb_md_type`	PBB domain types
`pbb_topology_type`	Topology type: peer ENNI or hierarchical ENNI

Member	Description
ma_isid_tree	Pointer to MA list for ISIDs
ma_te_sid_tre	Pointer to MA list for TE-SIDs which can exists on BCB or BEB
mip_creation	IEEE 802.1ag 2007: Section 17.5, Definitions for the Connectivity fault management MIB; Enumerated value indicating whether the management entity can create MHFs (MIP Half Function) for this Maintenance Domain
cfm_md_mhf_id_permission	IEEE 802.1ag 2007: Section 17.5, Definitions for the Connectivity fault management MIB; Enumerated value indicating what, if anything, to include in the Sender ID TLV (21.5.3) transmitted by MPs configured in this Maintenance Domain

```
struct cfm_md
{
  /* MD Name */
  u_char name[CFM_MD_NAME_LENGTH + 1];
  /* MD name type */
  u_int8_t name_type;
  /* This is used only for MAC domain name type
   * To store the two octet integer and the MAC address.
   * This is being added for ease of use while reading/writing
   * configuration. */
  s_int16_t mac_num;
  u_int8_t md_name_mac[MACADDR_LEN];
  /* MD Level */
  u_int32_t level;
  /* bridge */
  struct onmd_bridge *bridge;
  /* Pointer to MA list. One list accessed by vid, another using name*/
  struct avl_tree *ma_name_tree;
  struct avl_tree *ma_vid_tree;
  /* This would be the indexes for all the MAs.
   * Used for SNMP */
  struct bitmap *ma_index_bmp;
  u_int32_t md_index;
#if defined HAVE_CFM || defined HAVE_PBB_TE
  /* PBB domain types */
  enum cfm_pbb_md_type pbb_md_type;
#if (defined HAVE_I_BEB || defined HAVE_B_BEB)
  /* Topology type peer ENNI or hierarchical ENNI */
  u_int8_t pbb_topology_type;
#define CFM_PBB_MD_TOPO_TYPE_DEFAULT      1
#define CFM_PBB_MD_TOPO_TYPE_P_ENNI       2
#define CFM_PBB_MD_TOPO_TYPE_H_ENNI       3
  /* Pointer to MA list for ISIDs */
  struct avl_tree *ma_isid_tree;
#endif /* (HAVE_CFM && (defined HAVE_I_BEB || defined HAVE_B_BEB)) */
#endif /* HAVE_PBB_TE */
#if defined HAVE_CFM && defined HAVE_PBB_TE
  /* Pointer to MA list for TE-SID's which can exists on BCB or BEB */
```

```
  struct avl_tree *ma_te_sid_tree;
#endif /* HAVE_I_BEB && HAVE_B_BEB && HAVE_PBB_TE */
  /* IEEE 802.1ag 2007
   * Section 17.5, Definitions for the Connectivity fault management MIB */
  /*Enumerated value indicating whether the management entity can
    create MHFs (MIP Half Function) for this Maintenance Domain.*/
  u_int8_t mip_creation;
  /* Enumerated value indicating what, if anything, is to be
  included in the Sender ID TLV (21.5.3) transmitted by MPs
  configured in this Maintenance Domain. */
  u_int8_t cfm_md_mhf_id_permission;
};
```

cfm_tput_reception

This structure in `onmd/cfm/cfm_tst.h` defines throughput reception.

Member	Description
`mep`	Back Pointer to MEP structure
`tst_while`	Timer thread for periodically sending out TST frame
`tput_active`	Active throughput
`frame_rx_count`	Receive frame count
`receive_bytes`	Bytes received using the sequence number in the frame as offset

```
struct cfm_tput_reception
{
  /* Back Pointer to MEP structure */
  struct cfm_mep *mep;
  /* Timer thread for periodically sending out TST frame */
  struct thread *tst_while;
  u_int8_t tput_active;
  u_int8_t frame_rx_count;
  /* Storing the bytes received using the sequence number in the frame as offset */
  u_int16_t receive_bytes[CFM_MAX_TST_TPUT_FRAMES];
};
```

cfm_ma

This structure in `onmd/cfm/cfmd.h` defines a CFM maintenance association (MA).

Member	Description
`md`	Pointer to MD
`name_type`	MA name type

Member	Description
name	MA name as character string
ma_two_oct_integer	MA name as 2-octet integer
primary_vid_ma_name	For primary VID as MA name
vpn_id_ma_name	VPN-ID as MA name
mep_list	Pointer to MEP list
rmep_list	Pointer to RMEP list
vid	VLAN Identifier
isid	In Service domains, MA will be isid-based
te_instance	Backpointer to the te_instance
te_sid	TE service Identifier
event_list	Event list
index	Index
ma_index	MA index, used by SNMP
index_max	Maximum index
cci_enabled	Number of MEPs with CCM enabled
cc_aps_interval	Continue check interval
someRMEPCCMdefect	Remote MEP CCM defect
intf_status_count	Interface status count
port_status_count	Port status count
rmepCount	Remote MEP count
cci_interval	CCM Interval
mip_creation	IEEE 802.1ag 2007 Section 17.5 Definitions for the Connectivity Fault Management MIB: whether a MIP can be created
cfm_ma_id_permission	Enumerated value indicating what, if anything, to include in the Sender ID TLV
cfm_pbb_comp_id	The bridge component within the system to which the information in this entry applies
number_of_vids	Number of VIDs
row_status	For SNMP MIB
notify_evc_status	Whether EVC notification has been enabled for this VLAN

Member	Description
configured_rmep_count	Number of remote MEPs configured by user; needed to determine the MEP/MA connectivity status
converged_rmep_count	Number of remote MEPs that have converged; needed to determine the MEP/MA connectivity status
maid	Maintenance association identifier
maid_length	Actual length of the name without zero padding

```
struct cfm_ma
{
  /* Pointer to MD */
  struct cfm_md *md;
  /* MA name type */
  u_int8_t name_type;
  /* For Character string as MA Name */
  u_int8_t name[CFM_MA_NAME_LENGTH + 1];
  /* TWO OCTET INTEGER as MA-Name*/
  s_int16_t ma_two_oct_integer;
  /* For Primary VID as MA Name */
  u_int16_t primary_vid_ma_name;
  /* VPN-ID as MA-name */
  u_int8_t vpn_id_ma_name[7];
  /* Pointer to MEP list */
  struct avl_tree *mep_list;
  /* Pointer to RMEP list */
  struct avl_tree *rmep_list;
  u_int16_t vid;
#if defined HAVE_CFM && (defined HAVE_I_BEB || defined HAVE_B_BEB)
  /* In Service domains, MA will be isid based */
  u_int32_t isid;
#endif /* (HAVE_CFM && (defined HAVE_I_BEB || defined HAVE_B_BEB)) */
#if defined HAVE_CFM && defined HAVE_PBB_TE
#if defined HAVE_I_BEB && defined HAVE_B_BEB
  /* Backpointer to the te_instance */
  struct onm_pbb_te_instance *te_instance;
#endif /* HAVE_I_BEB && HAVE_B_BEB */
  u_int32_t te_sid;
#endif /* HAVE_CFM && HAVE_PBB_TE */
  struct list *event_list;
  u_int32_t index;
  /* MA index, used by SNMP */
  u_int32_t ma_index;
  u_int32_t index_max;
  /* Number of MEPs with CCM enabled */
  s_int16_t cci_enabled;
#if defined HAVE_CFM && defined HAVE_I_BEB && defined HAVE_B_BEB && \
  defined HAVE_PBB_TE
  u_int8_t cc_aps_interval;
```

```
#endif /* HAVE_CFM && HAVE_I_BEB && HAVE_B_BEB && HAVE_PBB_TE */
  s_int32_t someRMEPCCMdefect;
  s_int32_t intf_status_count;
  s_int32_t port_status_count;
  s_int16_t rmepCount;
  u_int8_t cci_interval;
  /* IEEE 802.1ag 2007, Section
   * 17.5 Definitions for the Connectivity Fault Management MIB */
  /* Value indicating whether a MIP can be created */
  enum cfm_mhf_creation mip_creation;
  /* Enumerated value indicating what, if anything, is to be
   * included in the Sender ID TLV */
  enum cfm_id_permission cfm_ma_id_permission;
  /* The bridge component within the system to which the information
   * in this entry applies */
  u_int16_t cfm_pbb_comp_id;
  /* Number of vids */
  u_int32_t number_of_vids;
  /* For SNMP MIB */
  u_int8_t row_status;
  /* Whether evc notification has been enabled on for this vlan or not. */
  u_int8_t notify_evc_status;
#define CFM_NOTIFY_EVC_STATUS_ENABLE   1
#define CFM_NOTIFY_EVC_STATUS_DISABLE  0
  /* The last evc state that has been given to ELMI */
  u_int8_t evc_status;
  /* Number of remote meps configured by user. This counter is needed
     to determine the MEP/MA connectivity status */
  u_int16_t configured_rmep_count;
  /* Number of remote meps that have converged. This counter is needed
     to determine the MEP/MA connectivity status */
  u_int16_t converged_rmep_count;
#ifdef HAVE_EOAM_HW_OFFLOAD
#define CFM_MAID_LENGTH 48
  u_int8_t maid[CFM_MAID_LENGTH];
  /* Actual length of the name without zero padding */
  u_int8_t maid_length;
#endif /* HAVE_EOAM_HW_OFFLOAD */
};
```

cfm_mep

This structure in `onmd/cfm/cfmd.h` defines a CFM maintenance end point (MEP).

Member	Description
`ma`	Pointer to MA
`mep_id`	MEP ID

Member	Description
level	MD Level
vid	MA's primary VID; used for rx
local_vid	Local VID on which the MEP sends frames (MEP's primary VID); this can be one of the MA's VIDs; used for tx
direction	UP or DOWN MEP
port	MEP port
mep_active	Admin state of port
mepTestingState	ITU-T Y1731 Section 7.7 Ethernet TST Signal: Whether MEP is configured for in-service or out-of-service testing
lck_tx_level	
tput_enable_reception	ITU-T Y1731 Section 8.0 OAM Functions for performance monitoring - Throughput measurement
priority	Priority defect value
highest_defect_priority	Highest priority defect value
lowest_alarm_priority	Lowest priority alarm value
ma_defect_indication	MA defect indication
highest_defect	Of all the defects currently present, which is the highest
cc	Pointer to continuity check data structure
lb	Pointer to loopback check data structure
lt	Pointer to linktrace data structure
unexpected_ltrs_rcvd	Unexpected link traces received
fa	Fault alarm
lm	ITU-T Y1731 Section 8.1.2 Single-Ended ETH-LM
ais	ITU-T Y1731 Section 7.4 ETH-AIS
dm_tx	ITU-T Y1731 Section 8.2.1 1-Way Delay measurement;
dm_rx	ITU-T Y1731 Section 8.2.1 1-Way Delay measurement
dmm	ITU-T Y1731 Section 8.2.2 2-Way Delay measurement
tst	ITU-T Y1731 Section 7.7 Ethernet TST Signal
lck	ITU-T Y1731 Section 7.6 Ethernet TST Signal
tput_rx	ITU-T Y1731 Section 8.3 Throughput measurement

Member	Description
mcc	ITU-T Y1731 Section 7.9 Ethernet Maintenance communication channel
exm	ITU-T Y1731 Section 7.10Ethernet experimental OAM
vsm	ITU-T Y1731 Section 7.9 Ethernet vendor-specific OAM
isid_up_mep_list	The BVLAN UP MEP has to be aware of the UP MEP in the service on the B-Component
isid	Service instance identifier
te_sid	TE service instance ID
esp_sa_mac	Encapsulating Security Payload source MAC address
esp_da_mac	Encapsulating Security Payload destination MAC address
pt_to_mpt_type	Whether a pt_to_mpt ESP
present_traffic	IEEE-802.1Qay-d4-5 clause 20.9: if the TE-SID is active path then this is set to 1
present_mm_loc	IEEE-802.1Qay-d4-5 clause 20.9
attribute	IEEE 802.1ag 2007
row_status	For SNMP MIB
pg	Pointer to 8031 protection group
ring	Pointer to 8032 ring node
is_ps	Whether MEP is configured for protocol switching
uni_mep	Flag to indicate mep as uni-meg and which is used intimate CFM uni meg status to NSM for UNI type
rmep_list	RMEP list maintained per MEP
presentRdi	Present Remote Defect Indications
stats	Statistics saved on MEP reconfig
routed_vlan	Whether this MEP is on a routed VLAN or switched VALN

```
struct cfm_mep
{
  /* Pointer to MA */
  struct cfm_ma *ma;
  /* MEPID */
  u_int32_t mep_id;
  /* MD Level */
  u_int8_t level;
  /* MA's primary-vid, for rx, this vid will be used. */
  u_int16_t vid;
  /* Local vid on which MEP will send frames (MEP's primary-vid)
   * This can be one of the MA's vids, for frame tx this vid will be used */
```

```
  u_int16_t local_vid;
  /* UP/DOWN MEP */
  u_char direction;
#define CFM_DOWN 1
#define CFM_UP   2
  /* MEP port */
  struct cfm_port *port;
  /* MEP Admin state of port */
  u_int8_t mep_active;
#ifdef HAVE_CFM_Y1731
  /* Section 7.7 Ethernet TST Signal of ITU-T Y1731 */
  /* Whether MEP is configured for in-service or out-of-service testing */
  u_int8_t mepTestingState;
  u_int8_t lck_tx_level;
  /* Section 8.0 OAM Functions for performance monitoring -
   *  Throughput measurement*/
  u_int8_t tput_enable_reception;
#endif /* HAVE_CFM_Y1731 */
  u_int32_t priority;
  u_int32_t highest_defect_priority;
  u_int32_t lowest_alarm_priority;
  u_char ma_defect_indication;
  /* Of all the defects currently present which is the highest one */
  u_int32_t highest_defect;
  /* Pointer to protocol Data Structures */
  /*Continuity Check */
  struct cfm_cc *cc;
  /* LoopBack */
  struct cfm_lb *lb;
  /* LinkTrace */
  struct cfm_lt *lt;
  u_int32_t unexpected_ltrs_rcvd;
  /* Fault Alarm */
  struct cfm_fa *fa;
#ifdef HAVE_CFM_Y1731
  /* For section 8.1.2 Single-ended ETH-LM of ITU-T Y1731 */
  struct cfm_lm *lm;
  /* For section 7.4 ETH-AIS of ITU-T Y1731 */
  struct cfm_ais *ais;
  /* Section 8.2.1 - 1-way Delay measurement */
  struct cfm_1dm_tx *dm_tx;
  /* Section 8.2.1 - 1-way Delay measurement */
  struct cfm_1dm_rx *dm_rx;
  /* Section 8.2.2 - 2-way Delay measurement  */
  struct cfm_dmm *dmm;
  /* Section 7.7 Ethernet TST Signal of ITU-T Y1731 */
  struct cfm_tst *tst;
  /* Section 7.6 Ethernet TST Signal of ITU-T Y1731 */
  struct cfm_lck *lck;
  /* Section 8.3 Throughput measurement of ITU-T Y1731 */
```

```
  struct cfm_tput_reception *tput_rx;
  /* Section 7.9 -Ethernet Maintenance communication channel */
  struct cfm_mcc *mcc;
  /* Section 7.10 -Ethernet experimental OAM */
  struct cfm_exm *exm;
  /* Section 7.9 -Ethernet vendor specific OAM */
  struct cfm_vsm *vsm;
#endif /* HAVE_CFM_Y1731 */
#if defined HAVE_CFM && (defined HAVE_I_BEB || defined HAVE_B_BEB)
  /* The BVLAN UP MEP has to be aware of the UP MEP in the service
   *  on the B-Component */
  struct list *isid_up_mep_list;
  u_int32_t isid;
#if defined HAVE_I_BEB && defined HAVE_B_BEB && defined HAVE_PBB_TE
  u_int32_t te_sid;
  u_char esp_sa_mac[MACADDR_LEN];
  u_char esp_da_mac[MACADDR_LEN];
  /* To mention whether its a pt_to_mpt ESP */
  u_int8_t pt_to_mpt_type;
  /* MEP variables as per the clause 20.9 in IEEE-802.1Qay-d4-5 */
  /* If the TE-SID is active path then this is set to 1 */
  bool_t present_traffic;
  bool_t present_mm_loc;
#endif /* HAVE_I_BEB && HAVE_B_BEB && HAVE_PBB_TE */
#endif /* (HAVE_CFM && (defined HAVE_I_BEB || defined HAVE_B_BEB)) */
  /* IEEE 802.1ag 2007 */
  struct cfm_mep_attributes attribute;
  /* For SNMP MIB */
  u_int8_t row_status;
  /* Pointer to 8031 Protection group */
#ifdef HAVE_G8031
  struct eps_protection_group  * pg;
#endif /* HAVE_G8031*/
  /* Pointer to 8032  */
#ifdef HAVE_G8032
  struct g8032_ring_node * ring;
#endif /* HAVE_G8032*/
#if defined HAVE_G8031 || defined HAVE_G8032 || defined HAVE_PBB_TE
  /* MEP configured for protocol switching */
  bool_t is_ps;
#endif /* HAVE_G8031 || HAVE_G8032 */
  u_int8_t uni_mep; //< Flag to indicate mep as uni-meg and which is used
                    //<  intimate CFM uni meg status to NSM for uni type
#ifdef HAVE_EOAM_HW_OFFLOAD
  /* RMEP list maintained per MEP */
  struct avl_tree *rmep_list;
  u_int8_t presentRdi;
#endif /* HAVE_EOAM_HW_OFFLOAD */
  /* This will keep the statistics saved on mep reconfig */
  struct cfm_statistics stats;
```



```
  /* Is this mep on a routed vlan or switched vlan */
  u_int8_t routed_vlan;
};
```

cfm_mip

This structure in `onmd/cfm/cfmd.h` defines a CFM maintenance intermediate point (MIP).

Member	Description
`port`	Bridge port
`level`	Link level
`vid`	VLAN identifier
`isid`	Service instance identifier
`valid`	Whether valid
`mip_ccm`	CCM variable
`lb`	Loopback variable
`lt`	Linktrace variable

```
struct cfm_mip
{
  struct cfm_port *port;
  u_int32_t level;
  u_int16_t vid;
#ifdef HAVE_B_BEB
  u_int32_t isid;
#endif /* HAVE_B_BEB */
  bool_t valid;
  /* MIP CCM Variables */
  struct cfm_ccm_frame *mip_ccm;
  /* MIP Loopback Variables */
  struct cfm_lb *lb;
  /* MIP LinkTrace Variable */
  struct cfm_lt *lt;
};
```

cfm_port

This structure in `onmd/cfm/cfmd.h` defines a CFM port.

Member	Description
ifp	Pointer to interface
bridge	Bridge
entity	Whether a configured entity
down_mep_list	Down Maintenance End Points list
up_mep_list	Up Maintenance End Points list
mip_list	Maintenance Intermediate Points list
mip_isid_list	MIP service instance list
mip	Maintenance Intermediate Points
server_mep	ITU-T Y1731 Section 7.4 ETH-AIS
ring_aps_node_list	List of ring nodes configured for this port
active_level_on_vid_list	List of active levels from which MIP is created on the bridge port. On each port, for each VID, there is a list of active levels. Each node here will be a VID and the array of active levels for that VID on this port.
link_lvl_bmp	Bitmap to check if a linklevel MEP is configured at a level
is_aggregated	LACP specific information: whether this is an aggregate port
ring_non_owner_if	The ring node non owner interface
is_shut	Whether the interface is shut

```
struct cfm_port
{
  struct interface *ifp;
  /* Bridge */
  struct onmd_bridge *bridge;
  /* Flag to indicate configured entity */
  u_int8_t entity;
#define CFM_PORT_UP_MEP_CONFIGURED        (1 << 0)
#define CFM_PORT_DOWN_MEP_CONFIGURED      (1 << 1)
#define CFM_PORT_UP_MHF_CONFIGURED        (1 << 2)
#define CFM_PORT_DOWN_MHF_CONFIGURED      (1 << 3)
  struct avl_tree *down_mep_list;
  struct avl_tree *up_mep_list;
  struct avl_tree *mip_list;
#ifdef HAVE_B_BEB
  struct avl_tree *mip_isid_list;
#endif /* HAVE_B_BEB */
/*  struct cfm_mip *mip; */
#ifdef HAVE_CFM_Y1731
  /* For section 7.4 ETH-AIS of ITU-T Y1731 */
```

```
  struct cfm_server_mep *server_mep;
#endif /* HAVE_CFM_Y1731 */
#ifdef HAVE_G8032
  /* This contains the list of ring node which are configured for this port */
   struct avl_tree *ring_aps_node_list;
#endif /* HAVE_G8032 */
  /* List of active level from which MIP will be created on the bridge port */
  /* On each port, for each vid there is a list of active level. */
  /* Each node here will be a vid and the array of active levels for that vid
   * on this port */
  struct avl_tree *active_level_on_vid_list;
  /* BMP to check if a LinkLevel MEP is configured at a level */
  u_int8_t link_lvl_bmp;
  /* LACP specific information */
#ifdef HAVE_LACPD
  /* Is this an aggregate port */
  u_int8_t is_aggregated;
#endif /* HAVE_LACPD */
#ifdef HAVE_G8032
  u_int8_t ring_non_owner_if;
#endif /*  HAVE_G8032 */
#ifdef HAVE_EOAM_HW_OFFLOAD
  /* Whether the interface is shut or no shut */
  u_int8_t is_shut;
#endif /* HAVE_EOAM_HW_OFFLOAD */
};
```

cfm_tst

This structure in `onmd/cfm/cfm_tst.h` holds information about a CFM test.

Member	Description
`mep`	Back pointer to MEP
`tst_while`	Timer thread for periodically sending out TST frame
`tst_data`	Pointer to form the frame and send
`tx_interval`	TX interval in seconds
`tst_active`	To release the CLI
`sequence_num`	Sequence number identifying TST frame
`destAddr`	Destination MAC address
`tst_frames_to_be_sent`	Total number of frames sent based on the time value for which TST was enabled
`tst_pattern`	Test pattern

Member	Description
tlv_length	Length of the TLV.
throughput_enable_sender	ITU-T Y1731 Section 8 Throughput: Denotes that MEP is sending TST for throughput measurement.

```
struct cfm_tst
{
  /* Back pointer to MEP */
  struct cfm_mep *mep;
  /* Timer thread for periodically sending out TST frame */
  struct thread *tst_while;
  /* Pointer to form the frame and send */
  u_int8_t *tst_data;
  /* TX interval in seconds */
  u_int8_t tx_interval;
  /* To release the CLI */
  u_int8_t tst_active;
   /* Sequence number identifying TST frame */
  u_int16_t sequence_num;
  /* Destination MAC address */
  u_int8_t destAddr[MACADDR_LEN];
  /* Total number of frames to be sent based on the time duration for which TST has been
enable */
  u_int8_t tst_frames_to_be_sent;
  /* Test pattern given from CLI */
  u_int8_t tst_pattern[CFM_MAX_TLV_TST_PATTERN_SIZE];
  /* Length of the TLV */
  u_int16_t tlv_length;
  /* For ITU-T Y1731 section 8, pg 24 - Throughput */
  u_int8_t throughput_enable_sender;
};
```

Command API

The functions in this section are called by the commands in the *Carrier Ethernet Command Reference*.

Function	Description
cfm_1dm_rx_enable	Enables and configures reception of one-way DM frames
cfm_add_ma	Adds a maintenance association
cfm_add_md	Adds a maintenance domain
cfm_add_mep	Adds a maintenance end point
cfm_add_mip	Adds a maintenance intermediate point
cfm_add_rmep	Adds a maintenance end point

Function	Description
cfm_add_secondary_vid	Adds a secondary VID
cfm_ais_disable	Disables AIS frames
cfm_ais_enable	Enables AIS frames
cfm_ais_set_tx_interval	Sets the transmission interval between AIS frames
cfm_auto_md_ma_mep_create	Configures an MD, MA, and MEP automatically
cfm_cc_mcast_disable	Disables point-to-point multicast Continuity Check Messaging
cfm_cc_mcast_enable	Enables point-to-point multicast Continuity Check Messaging
cfm_cc_unicast_disable	Disables point-to-point unicast Continuity Check Messaging
cfm_cc_unicast_enable	Enables point-to-point unicast Continuity Check Messaging
cfm_clear_default_md_level_entry	Resets the default MD level table
cfm_del_secondary_vid	Deletes a secondary VID
cfm_disable_dual_lm	Disables dual frame loss measurement
cfm_enable_dual_lm	Enables dual frame loss measurement
cfm_exm_send	Send experimental OAM frames
cfm_find_ma_by_isid	Finds a maintenance association
cfm_find_ma_by_vid	Finds a maintenance association
cfm_find_ma_mep_connectivity_status	Finds the connectivity status of an maintenance association
cfm_find_md_by_name	Finds a maintenance domain
cfm_find_mep	Finds a maintenance end point
cfm_lm_frame_count_sim	Starts the frame count simulator
cfm_mcc_send	Sends Maintenance Communication Channel frames
cfm_modify_default_md_level_entry	Modifies the default MD level table
cfm_remove_ma	Removes a maintenance association
cfm_remove_md	Removes a maintenance domain
cfm_remove_mep	Removes a maintenance end point
cfm_remove_mip	Removes a maintenance intermediate point
cfm_send_1dm	Sends one-way delay measurement frames
cfm_send_dmm	Sends two-way delay measurement frames
cfm_send_lmm	Sends Loss Measurement Messages
cfm_send_mcast_lb	Sends a multicast loopback request

Function	Description
cfm_send_ping	Sends a loopback message
cfm_send_ping2	Sends a loopback message
cfm_send_tst	Sends a test message
cfm_throughput_rx_enable	Enable throughput measurement
cfm_throughput_send_frame	Sends frame throughput messages
cfm_vsm_send	Sends vendor-specific OAM frames

Include Files

To use the functions in this chapter, you must include one or more of these files:

- `onmd/cfm/cfm_ais.h`
- `onmd/cfm/cfm_api.h`
- `onmd/cfm/cfm_dm.h`
- `onmd/cfm/cfm_dmm.h`
- `onmd/cfm/cfm_exm.h`
- onmd/cfm/cfm_ieee8021CfmMib.h
- `onmd/cfm/cfm_lm.h`
- `onmd/cfm/cfm_mcc.h`
- `onmd/cfm/cfm_pbb_api.h`
- `onmd/cfm/cfm_tst.h`
- `onmd/cfm/cfm_vsm.h`

cfm_1dm_rx_enable

This function is called by the `ethernet cfm 1dm receive` command to enable and configure reception of one-way DM frames for a MEP in a specific MD.

Frame delay and frame delay variation can performed in two ways:

- One-way DM sends an ETH_DM frame with the timestamp calculated as the time of transmission to a remote MEP in a point-to-point ME. The remote MEP then performs one-way delay (1DM) and/or one-way delay variation measurements (DVM).
- In two-way DM (DMM), a MEP sends frames with ETH-DM request information to its peer MEP and receives frames with ETH-DM reply information from its peer MEP to carry out two-way frame delay and two-way frame delay variation measurements.

Syntax

```
#include "cfm_dm.h"
struct cfm_1dm_rx *
cfm_1dm_rx_enable (struct onmd_bridge *bridge,
                   u_int8_t *md_name,
                   u_int16_t vid,
```

```
                u_int32_t mpid, u_int16_t duration)
```

Input Parameters

`bridge`	Bridge on which the MEP is configured
`md_name`	Name of the MD with which the MEP is associated
`vid`	VLAN ID <1-4094>
`mpid`	ID of the host MEP <1-8191>
`duration`	DM duration in seconds <5-60>

Output Parameters

None

Return Values

Pointer to the `cfm_1dm_rx` structure when the function succeeds

NULL when the function fails

cfm_add_ma

This function adds a maintenance association (MA) to a maintenance domain (MD).

Syntax

```
#include "cfm_api.h"
s_int32_t
cfm_add_ma (struct cfm_md *md, u_char *ma_name, u_int16_t vid, u_int8_t mip_permission,
            u_int8_t ma_name_type)
```

Input Parameters

`md`	Pointer to the MD
`ma_name`	Name of the MA
`vid`	VLAN ID <1-4094>
`mip_permission`	One of the following values from the `cfm_mhf_creation` enum in `onmd/cfm/cfmd.h`:
`CFM_DEF_MHF_NONE`	No MHFs can be created for this VID
`CFM_DEF_MHF_DEFAULT`	MHFs can be created on this VID on any bridge port through which this VID can pass
`CFM_DEF_MHF_EXPLICIT`	MHFs can be created for this VID only on bridge ports through which this VID can pass, and only if a MEP is created at some lower MD level
`CFM_DEF_MHF_DEFER`	The creation of MHFs is controlled by the MD variable dot1agCfmMaCompMhfCreation
`ma_name_type`	Format of the MA name; one of the following values from `onmd/cfm/cfmd.h`:
`CFM_MA_PRIMARY_VID`	Primary VLAN ID as the MA name

`CFM_MA_CHAR_STRING`

String as the MA name

`CFM_MA_TWO_OCTET_INTERGER`

Integer as MA name <0-65535>

`CFM_MA_RFC_2685_VPN_ID`

VPN ID as the MA name

Output Parameters

None

Return Values

CFM_API_ERR_INTERNAL when the MD does not exist

CFM_API_ERR_VLAN_NOT_FOUND when the VLAN is not configured

CFM_API_ERR_VID_CONFIGURED_FOR_VLAN when the VID is configured as a secondary VID

CFM_API_ERR_MA_EXISTS when the VLAN is already configured to some MA in the MD

CFM_API_ERR_MA_EXISTS when the MA already exists

CFM_API_ERR_MEM when memory allocation for the MA fails

CFM_API_ERR_INVALID_MA_NAME_LENGTH when the length of the MA name is not valid

CFM_API_ERR_INVALID_MA_NAME_TYPE when the type of the MA name is not valid

CFM_API_ERR_INVALID_MA_NAME when the name of the MA is not valid

CFM_API_ERR_HW_FAILURE when hardware fails

CFM_API_ERR_NONE when the function succeeds

cfm_add_md

This function adds a maintenance domain (MD) to a bridge.

Syntax

```
#include "cfm_api.h"
s_int32_t
cfm_add_md (struct onmd_bridge *bridge, u_char *md_name, u_int16_t level,
            u_int8_t mip_permission, u_int8_t md_name_type,
            struct cfm_md **ret_md)
```

Input Parameters

`bridge` Bridge

`md_name` Name of the MD

`level` Level of the MD

`mip_permission` Whether the MD can create MHFs (MIP Half Functions) for this VID at this MD level; one of these constants from `onmd/cfm/cfmd.h`:

`CFM_DEF_MHF_NONE`

No MHFs can be created for this VID

`CFM_DEF_MHF_DEFAULT`

MHFs can be created on this VID on any bridge port through which this VID can pass

`CFM_DEF_MHF_EXPLICIT`

MHFs can be created for this VID only on bridge ports through which this VID can pass, and only if a MEP is created at some lower MD level

`CFM_DEF_MHF_DEFER`

The creation of MHFs is controlled by the MD variable dot1agCfmMaCompMhfCreation

`md_name_type` One of these constants from `onmd/cfm/cfmd.h`:

`CFM_NO_MD_NAME`

No MD name present

`CFM_DNS_NAME`

DNS name

`CFM_MAC_ADDR_TWO_OCTET_INTEGER`

MAC address + 2-octet integer

`CFM_MD_CHAR_STRING`

Character string

Output Parameters

`ret_md` Pointer to the MD

Return Values

CFM_API_ERR_INVALID_ARG when the MD name or `ret_md` does not exist

CFM_API_ERR_BRIDGE_NOT_FOUND when the bridge does not exist

CFM_API_ERR_INTERNAL when the MD list does not exist

CFM_API_ERR_MD_NAME_NOT_NO_NAME when the MD name does not exist

CFM_API_ERR_INVALID_DNS_MD_NAME when the DNS name is invalid

CFM_API_ERR_INVALID_MAC when the `md_name` MAC address is not valid

CFM_API_ERR_INVALID_MD_NAME_TYPE when the MD name type is not CFM_MD_CHAR_STRING

CFM_API_ERR_MEM when memory allocation fails

CFM_API_ERR_MD_EXISTS when the MD already exists

CFM_API_ERR_NONE when the function succeeds

cfm_add_mep

This function adds a maintenance end point (MEP) to a bridge.

As per the MEF 20 standard, when a UNI MEP is configured, CCM is enabled by default. CCM can be disabled later.

Syntax

```
#include "cfm_api.h"
s_int32_t
cfm_add_mep (struct onmd_bridge *bridge, u_int16_t id, u_int8_t *md_name,
```

```
u_int16_t vid, u_char *dir, u_int16_t local_vid,
u_int8_t mep_active, u_int8_t uni_mep, struct interface *ifp)
```

Input Parameters

`bridge`	Bridge
`id`	ID of the MEP
`md_name`	MD name
`vid`	VLAN ID <1-4094>
`dir`	Whether an up (“up”) or down (any other string) MEP
`local_vid`	Local VID for MEP <1-4094>
`mep_active`	Whether active or inactive
`uni_mep`	Whether a UNI MEP (link MEP)
`ifp`	Pointer to the interface on which the MEP is configured

Output Parameters

None

Return Values

CFM_API_ERR_AGGREGATED_PORT when the port is aggregated

CFM_API_ERR_BRIDGE_NOT_FOUND when the bridge does not exist

CFM_API_ERR_IF_NOT_BOUND when the bridge port does not exist

CFM_API_ERR_IF_NOT_FOUND when the interface is not found

CFM_API_ERR_INTERNAL when the MD list is empty

CFM_API_ERR_MA_NOT_FOUND when the MA is not found

CFM_API_ERR_MD_NOT_FOUND when the MD is not found

CFM_API_ERR_UP_MEP_LINK_LEVEL CFM_API_ERR_PB_LINK_LEVEL_MEP when the PB link-level MEP is supported and the level is not 0

CFM_API_ERR_UNI_MEP_ON_WRONG_LEVE L when the MEP is a uni-MEP and the level is not 0

CFM_API_ERR_VLAN_NOT_FOUND when the VLAN is not found

CFM_API_ERR_SECONDARY_VID_NOT_FOR_VLAN when the VID is not found

CFM_API_ERR_SVID_ON_CEP when SVLAN is configured on CEP

CFM_API_ERR_PORT_MEP_EXISTS when the MEP port already exists

CFM_API_ERR_MA_MEP_EXISTS when the MEP already exists

CFM_API_ERR_MEM when memory allocation fails

RESULT_OK when the function succeeds

cfm_add_mip

This function adds a maintenance intermediate point (MIP) to a port.

Syntax

```
#include "cfm_api.h"
s_int32_t
cfm_add_mip (struct cfm_port *port, u_int16_t vid, u_int16_t level)
```

Input Parameters

`port`	Port
`vid`	VLAN ID<1-4094>
`level`	Maintenance level

Output Parameters

None

Return Values

Pointer to the cfm_mip struct when the function succeeds

NULL when the function fails

cfm_add_rmep

This function adds a static configuration for a remote MEP (RMEP) in a maintenance domain.

Syntax

```
#include "cfm_api.h"
s_int32_t
cfm_add_rmep (struct cfm_md *md, u_int32_t rmepid, u_int16_t vid,
              u_char *mac, u_int8_t flags)
```

Input Parameters

`md`	Pointer to the maintenance domain
`rmepid`	ID of the RMEP <1-8191>
`vid`	VLAN ID <1-4094>
`mac`	The MAC address of the RMEP
`flags`	One of these constants from `onmd/cfm/cfm_ccm.h`:

`CFM_RMEP_CONFIGURED`

`CFM_RMEP_LEARNT`

`CFM_RMEP_MAC_CONFIGURED`

Output Parameters

None

Return Values

CFM_API_ERR_INVALID_ARG when the MD does not exist

CFM_API_ERR_MA_NOT_FOUND when the MA is not found

CFM_API_ERR_NONE when the function succeeds

cfm_add_secondary_vid

This function is called by the command `ethernet cfm configure vlan bridge` to add a secondary VID to a VLAN.

Syntax

```
#include "cfm_api.h"
s_int32_t
cfm_add_secondary_vid (struct onmd_bridge *bridge,
                       u_int16_t primary_vid, u_int16_t vid)
```

Input Parameters

`bridge`	Bridge on which the VLAN is configured
`primary_vid`	Primary VLAN ID <1-4094>
`vid`	Secondary VLAN ID to add <1-4094>

Output Parameters

None

Return Values

CFM_API_ERR_VLAN_NOT_FOUND when the VLAN or secondary does not exist

CFM_API_ERR_BRIDGE_NOT_FOUND when the bridge is not found

CFM_API_ERR_PRIMARY_VID when the VID is already configured as a primary or secondary VID

CFM_API_ERR_VID_CONFIGURED_FOR_VLAN when the primary VID is not valid

CFM_API_ERR_VID_EXISTS when the VID already exists

CFM_API_ERR_MEM when inserting in the AVL tree fails

CFM_SUCCESS when the function succeeds

cfm_ais_disable

This function is called by the `ethernet cfm ais status` command when the `disable` option to disable AIS frames on the MEP in the MD.

Syntax

```
#include "cfm_ais.h"
s_int8_t
cfm_ais_disable (struct onmd_bridge *bridge, u_int32_t mepid,
                 u_int8_t *md_name,  u_int16_t vid, u_int8_t defect_type,
                 struct interface *ifp)
```

Input Parameters

`bridge`	Bridge
`mepid`	ID of the source MEP <1-8191>
`md_name`	Maintenance domain name
`vid`	VLAN ID <1-4094>

`defect_type` One of these constants from `onmd/cfm/cfm_ais.h`:

- `CFM_AIS_DEFECT_TYPE_ALL`
- `CFM_AIS_DEFECT_TYPE_LOC`
- `CFM_AIS_DEFECT_TYPE_MISMERGE`
- `CFM_AIS_DEFECT_TYPE_UNEXP_MEP`
- `CFM_AIS_DEFECT_TYPE_UNEXP_MEG_LVL`
- `CFM_AIS_DEFECT_TYPE_UNEXP_PERIOD`

`ifp` Pointer to the interface

Output Parameters

None

Return Value

CFM_FAILURE when the function fails

CFM_SUCCESS when the function succeeds

cfm_ais_enable

This function is called by the `ethernet cfm ais status` command when the `enable` option is used to configure Alarm Indicator Signaling (AIS) frames on the bridge in the MD.

Syntax

```
#include "cfm_ais.h"
s_int8_t
cfm_ais_enable (struct onmd_bridge *bridge, u_int32_t mepid, u_int8_t ais_type,
                u_int8_t *md_name, u_int8_t *rmac, s_int8_t destLevel,
                u_int16_t vid, u_int8_t defect_type, struct interface *ifp)
```

Input Parameters

`bridge` Bridge on which the host MEP exists

`mepid` ID of the source MEP <1-8191>

`ais_type` Unicast or multicast AIS frames

`md_name` Maintenance domain name

`rmac` For unicast frames, the remote MEP address in MAC format

`destLevel` Level at which the MEP exists

`vid` VLAN ID <1-4094>

`defect_type` One of these constants from `onmd/cfm/cfm_ais.h`:

- `CFM_AIS_DEFECT_TYPE_ALL`
- `CFM_AIS_DEFECT_TYPE_LOC`
- `CFM_AIS_DEFECT_TYPE_MISMERGE`
- `CFM_AIS_DEFECT_TYPE_UNEXP_MEP`
- `CFM_AIS_DEFECT_TYPE_UNEXP_MEG_LVL`
- `CFM_AIS_DEFECT_TYPE_UNEXP_PERIOD`

`ifp`	Pointer to the interface

Return Value

CFM_FAILURE when the bridge, MA, MD is not found

CFM_API_ERR_AIS_LINK_LEVEL_MEP when a link local MEP

CFM_API_ERR_AIS_DEFECT_COND_ALL when AIS defect condition is configured as ALL

CFM_SUCCESS when the function succeeds

cfm_ais_set_tx_interval

This function is called by the `ethernet cfm ais interval` command to set the transmission interval between AIS frames for the MEP in the MD.

Syntax

```
#include "cfm_ais.h"
s_int8_t
cfm_ais_set_tx_interval (struct onmd_bridge *bridge,
                         u_int8_t tx_interval, u_int32_t mepid,
                         u_int8_t *md_name, u_int16_t vid)
```

Input Parameters

`bridge`	ID of the bridge on which the MEP is configured
`tx_interval`	Transmission interval between AIS frames, either 1 or 60 seconds
`mepid`	ID of the source MEP <1-8191>
`md_name`	Name of the MD <1-4094>
`vid`	VLAN ID <1-4094>

Output Parameters

None

Return Values

CFM_FAILURE when the function fails

CFM_SUCCESS when the function succeeds

cfm_auto_md_ma_mep_create

This function is called by the `ethernet cfm automatic bridge` command to automatically configure CFM on the given bridge. When you call this function, there is no need to configure a maintenance domain (MD), a maintenance association (MA), and maintenance end points (MEPs) for the VLANs/I-SIDs associated with the bridge.

Syntax

```
#include "onmd/cfm/cfm_api.h"
s_int32_t
cfm_auto_md_ma_mep_create (struct onmd_bridge *bridge)
```

Input Parameters

`bridge` Bridge

Output Parameters

None

Return Values

CFM_API_ERR_BRIDGE_NOT_FOUND when the bridge is not found

CFM_API_ERR_NONE when the function succeeds

CLI_SUCCESS when the function succeeds

CFM_API_ERR_INVALID_ARG when there is an internal error

CFM_API_ERR_INTERNAL when there is an internal error

CFM_API_ERR_INVALID_MIP_PERMISSION when there is an internal error

CFM_API_ERR_MD_NAME_NOT_NO_NAME when there is an internal error

CFM_API_ERR_INVALID_DNS_MD_NAME when there is an internal error

CFM_API_ERR_INVALID_MAC when there is an internal error

CFM_API_ERR_INVALID_NAME_LEN_ITU_T when there is an internal error

CFM_API_ERR_INVALID_NAME_ITU_T when there is an internal error

CFM_API_ERR_INVALID_MD_NAME_TYPE when there is an internal error

CFM_API_ERR_MEM when there is an internal error

CFM_API_ERR_PBB_MD_TYPE when there is an internal error

CFM_API_ERR_PBB_TOPOLOGY_TYPE when there is an internal error

CFM_API_ERR_MEM when there is an internal error

CFM_API_ERR_MD_EXISTS when a maintenance domain already exists

-1 when there is an internal error

CFM_API_ERR_MA_NOT_FOUND when there is an internal error

CFM_API_ERR_ISID_NOT_CONFIGURED when there is an internal error

CFM_API_ERR_MA_EXISTS when a maintenance association already exists

CFM_API_ERR_VLAN_NOT_FOUND when there is an internal error

CFM_API_ERR_VID_CONFIGURED_FOR_VLAN when there is an internal error

CFM_API_ERR_INVALID_MA_NAME_TYPE when there is an internal error

cfm_cc_mcast_disable

This function is called by the `ethernet cfm cc multicast state` command with the `disable` option to disable point-to-point multicast CCM between a host MEP and a VID.

Syntax

```
#include "cfm_api.h"
s_int32_t
```

```
cfm_cc_mcast_disable (struct onmd_bridge *bridge,
                      u_int8_t *md_name, u_int16_t vid,
                      u_int32_t mepid, struct interface *ifp)
```

Input Parameters

`bridge`	Bridge on which the MEP exists
`md_name`	Maintenance domain name
`vid`	VLAN ID <1-4094>
`mepid`	ID of the source MEP <1-8191>
`ifp`	Pointer to the interface

Output Parameters

None

Return Value

CFM_BRIDGE_NOT_FOUND when the bridge is not found

CFM_API_ERR_MD_NOT_FOUND when the MD name is not found

when the MD list is not found

CFM_DOMAIN_NOT_FOUND when the MD is not found

CFM_API_ERR_IF_NOT_BOUND when the bridge port is not found for the interface

CFM_API_ERR_VLAN_NOT_FOUND when the VLAN is not found

CFM_API_ERR_VID_NOT_CONFIGURED_FOR_MA when the VID is not configured for MA

CFM_MA_NOT_FOUND when the MA is not found

CFM_API_ERR_MEP_NOT_FOUND when the MEP is not found

CFM_API_ERR_MEP_CCM_NOT_ENABLED when the CCM is not mcast enabled

RESULT_OK when the function succeeds

cfm_cc_mcast_enable

This function is called by the `ethernet cfm cc multicast state` command when the `enable` option is used to enable point-to-point multicast CCM between a host MEP and a VID on the MD name.

Syntax

```
#include "cfm_api.h"
s_int32_t
cfm_cc_mcast_enable (struct onmd_bridge *bridge,
                     u_int8_t *md_name, u_int16_t vid,
                     u_int32_t mepid, struct interface *ifp)
```

Input Parameters

`bridge`	Bridge on which the MEP exists
`md_name`	Maintenance domain name
`vid`	VLAN ID <1-4094>

`mepid`	ID of the source MEP <1-8191>
`ifp`	Pointer to the interface

Output Parameters

None

Return Value

CFM_BRIDGE_NOT_FOUND when the bridge is not found

CFM_API_ERR_MD_NOT_FOUND when the MD is not found

when the MD list is not found

CFM_DOMAIN_NOT_FOUND when the domain is not found

CFM_API_ERR_IF_NOT_BOUND when the bridge port is not found

CFM_API_ERR_VLAN_NOT_FOUND when the VLAN is not found

CFM_API_ERR_SECONDARY_VID when the primary VID is not valid

CFM_MA_NOT_FOUND when the MA is not found

CFM_API_ERR_MEP_NOT_FOUND when the MEP is not found

CFM_API_ERR_MEP_CCM_ALREADY_ENABLED when the CCM is already mcast- or unicast-enabled

CFM_API_MEP_NOT_ACTIVE when the MEP is not active

RESULT_OK when the function succeeds

cfm_cc_unicast_disable

This function is called by the `ethernet cfm unicast state` command when the option `disable` is used to disable point-to-point unicast Continuity Check Messaging (CCM) between the host MEP and a remote MEP.

Syntax

```
#include "cfm_api.h"
s_int32_t
cfm_cc_unicast_disable (struct onmd_bridge *bridge, u_int8_t *md_name,
                        u_int16_t vid, u_int32_t mpid, u_int32_t rmpid,
                        struct interface *ifp)
```

Input Parameters

`bridge`	Bridge on which the MEP exists
`md_name`	Name of the MD
`vid`	VLAN ID <1-4094>
`mpid`	ID of the host MEP <1-8191>
`rmpid`	ID of the remote MEP <1-8191>
`ifp`	Pointer to the interface

Return Value

CFM_BRIDGE_NOT_FOUND when the bridge is not found

when the MD list is not found

CFM_DOMAIN_NOT_FOUND when the MD is not found

CFM_API_ERR_IF_NOT_BOUND when the bridge port is not found

CFM_MA_NOT_FOUND when the MA is not found

CFM_MEP_NOT_FOUND when the MEP is not found

CFM_API_ERR_MEP_NOT_FOUND when the port interface index is not valid

CFM_API_ERR_MEP_CCM_NOT_ENABLED when the ccm is not unicast-enabled

CFM_RMEP_NOT_FOUND when the RMEP is not found

RESULT_OK when the function succeeds

cfm_cc_unicast_enable

This function is called by the `ethernet cfm unicast state` command when the option `enable` is used to enable point-to-point unicast Continuity Check Messaging (CCM) between the host MEP and a remote MEP. Unicast CCM frames are sent periodically to that MEP.

Syntax

```
#include "cfm_api.h"
s_int32_t
cfm_cc_unicast_enable (struct onmd_bridge *bridge, u_int8_t *md_name,
                       u_int16_t vid, u_int32_t mpid, u_int32_t rmpid,
                       struct interface *ifp)
```

Input Parameters

`bridge`	Bridge on which the MEP exists
`md_name`	Name of the MD
`vid`	VLAN ID <1-4094>
`mpid`	ID of the host MEP <1-8191>
`rmpid`	ID of the remote MEP <1-8191>
`ifp`	Pointer to the interface

Return Value

CFM_BRIDGE_NOT_FOUND when the bridge is not found

when the MD list is not found

CFM_DOMAIN_NOT_FOUND when the MD is not found

CFM_API_ERR_IF_NOT_BOUND when the bridge port is not found

CFM_API_ERR_VLAN_NOT_FOUND when the VLAN is not found

CFM_API_ERR_SECONDARY_VID when the primary VID is not valid

CFM_MA_NOT_FOUND when the MA is not found

CFM_MEP_NOT_FOUND when the MEP is not found

CFM_API_ERR_MEP_NOT_FOUND when the port MEP is not found

CFM_API_MEP_NOT_ACTIVE when the MEP is not active

CFM_API_ERR_MEP_CCM_ALREADY_ENABLED when the CCM is already enabled

CFM_RMEP_NOT_FOUND when the RMEP is not found

CFM_RMEP_MAC_NOT_FOUND when the RMEP MAC is invalid

RESULT_OK when the function succeeds

cfm_clear_default_md_level_entry

This function resets the default MD level table.

Syntax

```
#include "cfm_api.h"
s_int32_t
cfm_clear_default_md_level_entry (struct onmd_bridge *bridge, u_int16_t vid,
                                  u_int32_t isid)
```

Input Parameters

`bridge`	Bridge
`vid`	VLAN ID <1-4094>
`isid`	ISID for service MIPs

Output Parameters

None

Return Values

CFM_API_ERR_BRIDGE_NOT_FOUND when the bridge is not found

CFM_API_ERR_DEF_MD_TBL_ENTRY_NOT_FOUND when the default MD level entry is not found

CFM_SUCCESS when the function succeeds

cfm_del_secondary_vid

This function deletes a secondary VID from a VLAN.

Note: A primary VID that identifies a VLAN cannot be deleted using this function.

Syntax

```
#include "cfm_api.h"
s_int32_t
cfm_del_secondary_vid (struct onmd_bridge *bridge, struct onm_vlan *vlan,
                       u_int16_t vid)
```

Input Parameters

`bridge`	Bridge on which the VLAN is configured
`vlan`	Pointer to the VLAN
`vid`	Secondary VLAN ID to delete <1-4094>

Output Parameters

None

Return Values

CFM_API_ERR_VLAN_NOT_FOUND when the VLAN is not found

CFM_API_ERR_BRIDGE_NOT_FOUND when the bridge is not found

CFM_API_ERR_SECONDARY_VID_NOT_FOR_VLAN when the secondary VID is not mapped to the primary VID

CFM_SUCCESS when the function succeeds

cfm_disable_dual_lm

This function disables dual frame loss measurement from being encoded in the CCMs transmitted from this MEP. In addition, if it is disabled, then even on receiving a CCM frame with ETH-LM information from its peer MEP, the host MEP does not compute loss measurement, discarding the frame count information.

Syntax

```
#include "cfm_lm.h"
s_int32_t
cfm_disable_dual_lm (struct onmd_bridge *bridge, u_int16_t mep_id,
                     u_int16_t rmep_id, u_int8_t *md_name, u_int16_t vid,
                     struct interface *ifp)
```

Input Parameters

`bridge`	Bridge on which the MEP exists
`mep_id`	ID of the MEP <1-8191>
`rmep_id`	ID of the RMEP <1-8191>
`md_name`	Name of the Maintenance Domain
`vid`	VLAN ID <1-4094>
`ifp`	Interface pointer

Return Value

CFM_BRIDGE_NOT_FOUND when the bridge is not found

when the MD list is not found

CFM_DOMAIN_NOT_FOUND when the MD is not found

CFM_API_ERR_VLAN_NOT_FOUND when the VLAN is not found

CFM_API_ERR_SECONDARY_VID when the primary VID is not valid

CFM_MA_NOT_FOUND when the MA is not found

CFM_MEP_NOT_FOUND when the MEP is not found

CFM_API_MEP_NOT_ACTIVE when the MEP is not active

CFM_API_ERR_LM_CCM_UCAST when the CCM is not unicast-enabled

CFM_RMEP_NOT_FOUND when the RMEP is not found

CFM_API_ERR_LM_DISABLED when loss measurement is already disabled

RESULT_OK when the function succeeds

cfm_enable_dual_lm

This function enables dual loss measurement being encoded in the CCMs transmitted rom this MEP. Dual-ended ETH-LM can be enabled only between MEPs with unicast CCM running between them.

Syntax

```
#include "cfm_lm.h"
s_int32_t
cfm_enable_dual_lm (struct onmd_bridge *bridge, u_int16_t mep_id, u_int16_t rmep_id,
u_int8_t *md_name, u_int16_t vid, struct interface *ifp)
```

Input Parameters

`bridge`	Bridge on which the MEP exists
`ifp`	Interface pointer
`md_name`	Name of the Maintenance Domain
`mepid`	ID of the MEP <1-8191>
`rmepid`	ID of the RMEP <1-8191>

Return Value

CFM_BRIDGE_NOT_FOUND when the bridge is not found

when the MD list is not found

CFM_DOMAIN_NOT_FOUND when the MD is not found

CFM_API_ERR_VLAN_NOT_FOUND when the VLAN is not found

CFM_API_ERR_SECONDARY_VID when the primary VID is not valid

CFM_MA_NOT_FOUND when the MA is not found

CFM_MEP_NOT_FOUND when the MEP is not found

CFM_API_MEP_NOT_ACTIVE when the MEP is not active

CFM_API_ERR_LM_CCM_UCAST when the CCM is not unicast-enabled

CFM_RMEP_NOT_FOUND when the RMEP is not found

CFM_API_ERR_LM_ENABLED when loss measurement is already enabled

RESULT_OK when the function succeeds

cfm_exm_send

This function is called by the `ethernet cfm exm` to send experimental (EXM) OAM frames.

Syntax

```
#include "cfm_exm.h"
s_int32_t
cfm_exm_send (struct onmd_bridge *bridge, u_int8_t *md_name,
              u_int16_t vid, u_int32_t mepid, u_int32_t rmepid
```

Input Parameters

`bridge`	Bridge

`md_name`	Maintenance domain name
`mepid`	ID of the source MEP <1-8191>
`vid`	VLAN ID <1-4094>
`rmepid`	ID of the remote MEP <1-8191>

Return Value

CFM_BRIDGE_NOT_FOUND when the bridge is not found

when the MD list is not found

CFM_DOMAIN_NOT_FOUND when the MD is not found

CFM_MA_NOT_FOUND when the MA is not found

CFM_MEP_NOT_FOUND when the MEP is not found

CFM_API_MEP_NOT_ACTIVE when the MEP is not active

CFM_MEMORY_NOT_ALLOCATED when memory allocation fails

CFM_RMEP_NOT_FOUND when the RMEP is not found

RESULT_OK when the function succeeds

cfm_find_ma_by_isid

This function finds a specific maintenance association.

Syntax

```
#include "cfm_pbb_api.h"
struct cfm_ma *
cfm_find_ma_by_isid (struct cfm_md *md, u_int32_t isid)
```

Input Parameters

`md`	Name of the MD
`isid`	Service entity identifier

Output Parameters

None

Return Values

Pointer to the `cfm_ma` structure when the function succeeds

NULL when not found

cfm_find_ma_by_vid

This function finds a specific maintenance association.

Syntax

```
#include "cfm_api.h"
struct cfm_ma *
cfm_find_ma_by_vid (struct cfm_md *md, u_int16_t vid)
```

Input Parameters

`md`	Name of the MD
`vid`	VLAN ID <1-4094>

Output Parameters

None

Return Values

Pointer to the `cfm_ma` structure when the function succeeds

NULL when not found

cfm_find_ma_mep_connectivity_status

This function finds the connectivity status of an MA and all the MEPS in the MA.

Syntax

```
#include "cfm_api.h"
enum cfm_ma_status
cfm_find_ma_mep_connectivity_status (struct cfm_ma *ma, u_int8_t *pbuf)
```

Input Parameters

`ma`	Pointer to the MA
`pbuf`	Pointer to the buffer where the connectivity status of MEPs is copied.

Output Parameters

None

Return Values

CFM_MA_STATUS_NOT_ACTIVE when the MA is not active

CFM_MA_STATUS_ACTIVE when the MA is active

CFM_MA_STATUS_PARTIALLY_ACTIVE when the MA is partially active

CFM_MA_STATUS_ERROR when the function fails

cfm_find_md_by_name

This function finds a maintenance domain name from the MD list.

Syntax

```
#include "cfm_api.h"
struct cfm_md *
cfm_find_md_by_name (struct list *md_list, u_char *md_name)
```

Input Parameters

`md_list`	Pointer to the list of MDs
`md_name`	Maintenance domain name

Output Parameters

None

Return Values

Pointer to the `cfm_md` structure when the function succeeds

NULL when not found

cfm_find_mep

This function finds a specific maintenance end point.

Syntax

```
#include "cfm_api.h"
struct cfm_mep *
cfm_find_mep (struct cfm_ma *ma, u_int32_t mep_id)
```

Input Parameters

`ma`	Pointer to the MA
`mep_id`	ID of the MEP

Output Parameters

None

Return Values

Pointer to the `cfm_mep` structure when the function succeeds

NULL when not found

cfm_lm_frame_count_sim

This function is called by the `ethernet cfm lmm counter` command to start the frame count simulator, which generates a counter simulating a hardware counter for incoming and outgoing loss measurement (LM) frames.

Syntax

```
#include "cfm_lm.h"
void
cfm_lm_frame_count_sim()
```

Input Parameters

None

Return Value

None

cfm_mcc_send

This function is called by the `ethernet cfm mcc tx` command to send Maintenance Communication Channel (MCC) frames.

Syntax

```
#include "cfm_mcc.h"
s_int32_t
cfm_mcc_send (struct onmd_bridge *bridge, u_int8_t *md_name,
              u_int16_t vid, u_int32_t mepid,
              u_int32_t rmepid, u_int8_t frame_type)
```

Input Parameters

`bridge`	Bridge on which the MEP is located
`md_name`	Maintenance domain name
`vid`	VLAN ID <1-4094>
`mepid`	ID of the source MEP <1-8191>
`rmepid`	For unicast frames, the ID of the remote MEP <1-8191>
`frame_type`	Unicast or multicast frames

Output Parameters

None

Return Value

CFM_BRIDGE_NOT_FOUND when the bridge is not found

when the MD list is not found

CFM_DOMAIN_NOT_FOUND when the MD is not found

CFM_MA_NOT_FOUND when the MA is not found

CFM_MEP_NOT_FOUND when the MEP is not found

CFM_API_MEP_NOT_ACTIVE when the MEP is not active

CFM_MEMORY_NOT_ALLOCATED when memory allocation fails

CFM_RMEP_NOT_FOUND when the RMEP is not found

RESULT_OK when the function succeeds

cfm_modify_default_md_level_entry

This function modifies the default MD level table.

Syntax

```
#include "cfm_api.h"
s_int32_t
cfm_modify_default_md_level_entry (struct onmd_bridge *bridge,
                                   u_int16_t vid, u_int32_t isid,
                                   u_int8_t level, s_int8_t permission)
```

Input Parameters

`bridge` Bridge on which the vlan is configured

`vid` VLAN ID <1-4094>

`isid` ISID for service MIPs

`level` Level at which the entry is to be added <0-7>

`permission` Whether the MD can create MHFs (MIP Half Functions) for this VID at this MD level; one of these constants from `onmd/cfm/cfmd.h`:

`CFM_DEF_MHF_NONE`
No MHFs can be created for this VID

`CFM_DEF_MHF_DEFAULT`
MHFs can be created on this VID on any bridge port through which this VID can pass

`CFM_DEF_MHF_EXPLICIT`
MHFs can be created for this VID only on bridge ports through which this VID can pass, and only if a MEP is created at some lower MD level

Output Parameters

None

Return Values

CFM_API_ERR_BRIDGE_NOT_FOUND when the bridge is not found

CFM_API_ERR_DEF_MD_TBL_ENTRY_NOT_FOUND when the default MD level entry is not found

CFM_SUCCESS when the function succeeds

cfm_remove_ma

This function removes a maintenance association from a specific MD.

Syntax

```
#include "cfm_api.h"
s_int32_t
cfm_remove_ma (struct cfm_md *md, u_char *ma_name, u_int16_t vid,
               u_int8_t mip_permission)
```

Input Parameters

`md` Pointer to the MD with which the MA is associated

`ma_name` Name of the MA

`vid` VLAN ID <1-4094>

`mip_permission` Whether the MD can create MHFs (MIP Half Functions) for this VID at this MD level; one of these constants from `onmd/cfm/cfmd.h`:

`CFM_DEF_MHF_NONE`
No MHFs can be created for this VID

`CFM_DEF_MHF_DEFAULT`
MHFs can be created on this VID on any bridge port through which this VID can pass

`CFM_DEF_MHF_EXPLICIT`

MHFs can be created for this VID only on bridge ports through which this VID can pass, and only if a MEP is created at some lower MD level

`CFM_DEF_MHF_DEFER`

The creation of MHFs is controlled by the MD variable dot1agCfmMaCompMhfCreation

Output Parameters

None

Return Values

CFM_API_ERR_INTERNAL when the MD is not found

CFM_API_ERR_VLAN_NOT_FOUND? when the vlan is not found

CFM_API_ERR_MA_NOT_FOUND when the MA is not found

CFM_API_ERR_MIP_PERMISSION MIP permission is not found

CFM_API_ERR_HW_FAILURE in case of hardware failure

CFM_API_ERR_NONE when the function succeeds

cfm_remove_md

This function removes a maintenance domain from a bridge.

Syntax

```
#include "cfm_api.h"
s_int32_t
cfm_remove_md (struct onmd_bridge *bridge, u_char *md_name, u_int16_t level,
               u_int8_t mip_permission)
```

Input Parameters

`bridge` Bridge

`md_name` Maintenance domain name

`level` Level of the maintenance domain

`mip_permission` Whether the MD can create MHFs (MIP Half Functions) for this VID at this MD level; one of these constants from `onmd/cfm/cfmd.h`:

`CFM_DEF_MHF_NONE`

No MHFs can be created for this VID

`CFM_DEF_MHF_DEFAULT`

MHFs can be created on this VID on any bridge port through which this VID can pass

`CFM_DEF_MHF_EXPLICIT`

MHFs can be created for this VID only on bridge ports through which this VID can pass, and only if a MEP is created at some lower MD level

`CFM_DEF_MHF_DEFER`

The creation of MHFs is controlled by the MD variable dot1agCfmMaCompMhfCreation

Output Parameters

None

Return Values

CFM_API_ERR_BRIDGE_NOT_FOUND when the bridge is not found

CFM_API_ERR_INTERNAL when the MD list is not found

CFM_API_ERR_NONE when the function succeeds

cfm_remove_mep

This function removes a maintenance end point from a specified MD and VLAN.

Syntax

```
#include "cfm_api.h"
s_int32_t
cfm_remove_mep (struct onmd_bridge *bridge, u_int16_t id, u_int8_t *md_name,
                u_int16_t vid, u_char *dir, u_int16_t local_vid,
                struct interface *ifp)
```

Input Parameters

`bridge`	Bridge
`id`	ID of the MEP
`md_name`	Maintenance domain name
`vid`	VLAN ID <1-4094>
`dir`	Direction of flow on the MEP
`local_vid`	Local VLAN ID <1-4094>
`ifp`	Pointer to the interface

Output Parameters

None

Return Values

CFM_API_ERR_BRIDGE_NOT_FOUND when the bridge is not found

CFM_API_ERR_IF_NOT_FOUND when the interface is not found

CFM_API_ERR_MD_NOT_FOUND when the MD is not found

CFM_API_ERR_INTERNAL when the MD list is not found

CFM_API_ERR_MA_NOT_FOUND when the MA is not found

CFM_API_ERR_MEP_NOT_FOUND when the MEP is not found

CFM_SUCCESS when the function succeeds

cfm_remove_mip

This function removes a MIP on a port.

Syntax

```
#include "cfm_api.h"
void
cfm_remove_mip (struct cfm_port *port, struct cfm_mip *mip)
```

Input Parameters

`port`	Port
`mip`	MIP

Output Parameters

None

Return Values

None

cfm_send_1dm

This function is called by the command `ethernet cfm 1dm` parameter to send one-way delay measurement frames on for a MEP in a specific MD.

Syntax

```
#include "cfm_dm.h"
struct cfm_1dm_tx *
cfm_send_1dm (struct onmd_bridge *bridge, u_int8_t *md_name,
              u_int16_t vid, u_int32_t mpid, u_int32_t rmpid,
              u_int16_t duration, u_int8_t tx_interval,
              u_int8_t frame_type)
```

Input Parameters

`bridge`	Bridge
`md_name`	Name of the MD
`vid`	VLAN ID <1-4094>
`mpid`	ID of the host MEP <1-8191>
`rmip`	ID of the remote MEP <1-8191>
`duration`	DM duration in the range of <5-50> seconds
`interval`	DM transmission interval
`frame_type`	Whether unicast or multicast

Output Parameters

None

Return Values

Pointer to the `cfm_1dm_tx` structure when the function succeeds

NULL when not sent

cfm_send_dmm

This function is called by the command `ethernet cfm dmm` to send two-way delay measurement frames on for a MEP in a specific MD.

Syntax

```
#include "cfm_dmm.h"
struct cfm_dmm *
cfm_send_dmm (struct onmd_bridge *bridge, u_int8_t *md_name,
              u_int16_t vid, u_int32_t mpid, u_int32_t rmpid,
              u_int16_t duration, u_int8_t tx_interval,
              u_int8_t frame_type)
```

Input Parameters

`bridge`	Bridge on which the MEP is configured
`md_name`	Name of the MD with which the MEP is associated
`vid`	VLAN ID <1-4094>
`mpid`	ID of the host MEP <1-8191>
`rmip`	ID of the remote MEP <1-8191>
`duration`	Duration in seconds <5-50>
`interval`	DM transmission interval
`frame_type`	Whether unicast or multicast

Output Parameters

None

Return Values

Pointer to the `cfm_dmm` structure when the function succeeds

NULL when not sent

cfm_send_lmm

This function is called by the `ethernet cfm lmm unicast|multicas`t command and is used to send Loss Measurement Messages between a source and a remote MEP.

Syntax

```
#include "cfm_lm.h"
struct cfm_lm *
cfm_send_lmm (struct onmd_bridge *bridge, u_int32_t mepid,
              u_int32_t rmepid, u_int8_t lm_type,
              u_int16_t duration, u_int8_t tx_interval,
              u_int8_t *md_name, u_int16_t vid)
```

Input Parameters

`bridge`	ID of the bridge on which the MEP is configured
`mepid`	ID of the source MEP <1-8191>

`rmepid`	ID of the remote MEP <1-8191>
`lm_type`	Type of LM messages to be sent
`duration`	Duration in the range of <5-60>
`tx_interval`	Transmission interval between LMM frames
`md_name`	Name of the MD
`vid`	VLAN ID <1-4094>

Output Parameters

None

Return Values

Pointer to the cfm_lm structure when the function succeeds

NULL when not sent

cfm_send_mcast_lb

This function is called by the `ping ethernet multicast` command to send a multicast loopback request.

Note: This function is only available if you have a ZebOS-XP built with ITU Y.1731 support.

Syntax

```
#include "cfm_api.h"
struct cfm_lb *
cfm_send_mcast_lb (struct onmd_bridge *bridge, u_char *md_name, u_int32_t mepid,
                   u_int8_t tlv_type, u_int16_t tlv_indicator,
                   s_int8_t level, u_int16_t vid, u_int8_t recursive_mcast)
```

Input Parameters

`bridge`	Bridge on which the host MEP exists
`md_name`	Name of the Maintenance Domain
`mepid`	ID of the MEP <1-8191>
`tlv_type`	Type of TLV (3 for data or 32 for test frames)
`tlv_indicator`	Indicates that a TLV is passed in the command
`level`	MD level
`vid`	VLAN ID <1-4094>
`recursive_mcast`	Whether to send multicast frames recursively (maximum of five frames)

Return Value

Pointer to the `struct cfm_lb` structure when the function succeeds

cfm_send_ping

This function sends loopback messages to a remote MEP.

This function has two signatures that are identical except the first has additional parameters. Use the second signature if you have a ZebOS-XP built with ITU Y.1731 support, otherwise use the first signature.

Syntax

```
#include "cfm_api.h"
s_int32_t
cfm_send_ping(struct onmd_bridge *bridge, u_int32_t rmepid,
              u_char *md_name, u_int8_t level, u_int16_t vid,
              u_int16_t source_mep_id, u_int8_t tlv_type,
              u_int16_t tlv_indicator, u_int8_t tx_interval,
              u_int8_t tx_lbm_count, u_int8_t reply_timeout,
              struct cfm_lb **ret_lb)

#include "cfm_api.h"
struct cfm_lb *
cfm_send_ping(struct onmd_bridge *bridge, u_int32_t mepid, u_int32_t rmepid,
              u_int8_t *md_name, u_int8_t tlv_type, u_int16_t tlv_indicator,
              s_int8_t level, u_int16_t vid)
```

Input Parameters

`bridge`	Bridge on which the ping is to be executed
`mepid`	ID of the host MEP <1-8191>
`rmepid`	ID of the remote MEP <1-8191>
`md_name`	Name of the MD to which the host MEP is associated
`tlv_type`	Type of TLV information requested

`tlv_indicator`	Test pattern: one of these values from the `test_pattern_indicator` enum in `onmd/cfm/cfm_lb.h`:
	`TEST_TLV_PATTERN_1`
	`TEST_TLV_PATTERN_2`
	`TEST_TLV_PATTERN_3`
	`TEST_TLV_PATTERN_4`

`level`	MD level
`vid`	VLAN ID <1-4094>
`tx_interval`	Transmission interval
`tx_lbm_count`	Time out after loopback transmission
`reply_timeout`	Time out for loopback reply
`ret_lb`	Pointer to cfm_lb struct

Output Parameters

None

Return Values

Without ITU Y.1731 support:

CFM_API_ERR_BRIDGE_NOT_FOUND when the bridge does not exist

when the MD list is not found

CFM_API_ERR_MD_NOT_FOUND when the MD is not found

CFM_API_ERR_VLAN_NOT_FOUND when the VLAN is not found

CFM_API_ERR_MA_NOT_FOUND when the MA is not found

CFM_API_ERR_MEP_NOT_FOUND when the MEP is not found

CFM_API_MEP_NOT_ACTIVE when the MEP is not active

CFM_API_ERR_LB_NOT_FOUND when the loopback for the MEP is NULL

CFM_API_LBI_ALREADY_ACTIVE when loopback is already set

CFM_API_ERR_RMEP_NOT_FOUND when the remote MEP cannot be found

CFM_API_ERR_INVALID_TLV_INDICATOR when `tlv_indicator` is invalid

CFM_API_ERR_INVALID_REPLY_INT when `reply_timeout` is greater than the `tx_interval`

CFM_API_ERR_NONE when the function succeeds

With ITU Y.1731 support:

Pointer to the `cfm_lb` structure when the function succeeds

NULL when nothing is returned

cfm_send_ping2

This function is called by the `ping ethernet mac` command to send a loopback message from the MAC address of the source MEP.

This function has two signatures that are identical except the first has additional parameters. Use the second signature if you have a ZebOS-XP built with ITU Y.1731 support, otherwise use the first signature.

Syntax

```
#include "cfm_api.h"
s_int32_t
cfm_send_ping2(struct onmd_bridge *bridge, u_char *md_name,
               u_int8_t level, u_int16_t vid, u_char *mac,
               u_int16_t source_mep_id, u_int8_t tlv_type,
               u_int16_t tlv_indicator, u_int8_t tx_interval,
               u_int8_t tx_lbm_count, u_int8_t reply_timeout,
               struct cfm_lb **ret_lb)

#include "cfm_api.h"
struct cfm_lb *
cfm_send_ping2(struct onmd_bridge *bridge, u_char *md_name,
               s_int8_t level, u_int16_t vid, u_char *mac,
               u_int16_t source_mep_id, u_int8_t tlv_type, u_int16_t tlv_indicator)
```

Input Parameters

`bridge`	Bridge on which the ping request is executed
`md_name`	Name of the MD to which the host MEP is associated
`level`	MD level

`vid`	VLAN ID <1-4094>
`mac`	MAC address of the source MEP
`source_mep_id`	ID of the source MEP
`tlv_type`	TLV type
`tlv_indicator`	Test pattern: one of these values from the `test_pattern_indicator` enum in `onmd/cfm/cfm_lb.h`:
	`TEST_TLV_PATTERN_1`
	`TEST_TLV_PATTERN_2`
	`TEST_TLV_PATTERN_3`
	`TEST_TLV_PATTERN_4`
`tx_interval`	Transmission interval
`tx_lbm_count`	Time out after loopback transmission
`reply_timeout`	Time out for loopback reply
`ret_lb`	Pointer to cfm_lb struct

Return Value

Without ITU Y.1731 support:

CFM_API_ERR_BRIDGE_NOT_FOUND when the bridge does not exist

when the MD list is not found

CFM_API_ERR_MEP_MCAST_MAC_NOT_ALLOWED when a MAC address is not allowed for the platform

CFM_API_ERR_MD_NOT_FOUND when the MD is not found

CFM_API_ERR_VID_NOT_CONFIGURED_FOR_MA when the VID is not configured for MA

CFM_API_ERR_MA_NOT_FOUND when the MA is not found

CFM_API_ERR_MEP_NOT_FOUND when the MEP is not found

CFM_API_MEP_NOT_ACTIVE when the MEP is not active

CFM_API_ERR_LB_NOT_FOUND when the loopback for the MEP is NULL

CFM_API_LBI_ALREADY_ACTIVE when loopback is already set

CFM_API_ERR_MEP_AMBIGUOUS when `source_mep_id` is ambiguous

CFM_API_ERR_INVALID_TLV_INDICATOR when `tlv_indicator` is invalid

CFM_API_ERR_INVALID_REPLY_INT when `reply_timeout` is greater than the `tx_interval`

CFM_API_ERR_NONE when the function succeeds

With ITU Y.1731 support:

Pointer to the `cfm_lb` structure when the function succeeds

NULL when the function fails

cfm_send_tst

This function is called by the command `ethernet cfm tst level vlan` to configure unicast or multicast test frames.

If CFM is configured for in-service testing, this function sends a test (TST) frame alone; if CFM is configured for out-of-service testing, this function sends a TST frame and a lock (LCK) frame.

Syntax

```
#include "cfm_tst.h"
struct cfm_tst *
cfm_send_tst (struct onmd_bridge *bridge, u_int32_t mepid, u_int32_t rmepid,
              u_int8_t frame_type, u_int8_t pat_indicator, u_int8_t duration,
              s_int8_t tx_interval, u_int8_t *md_name, u_int16_t vid,
              u_int8_t recursive, u_int8_t lck_tx_interval,
              u_int32_t lck_rmepid, u_int8_t lck_frame_type)
```

Input Parameters

`bridge`	Bridge on which the host MEP exists
`mepid`	ID of the source MEP <1-8191>
`rmepid`	For unicast test frames, the remote MEP ID <1-8191>
`frame_type`	Type of TST frame to send, unicast or multicast
`pat_indicator`	Test pattern
`duration`	Duration in the range of <5-60> seconds to send TST frames
`tx_interval`	Transmission interval between TST frames
`md_name`	Maintenance domain name
`vid`	VLAN ID <1-4094>
`recursive`	Send continuous frames at the interval specified
`lck_tx_interval`	Transmission interval between LCK frames
`lck_rmepid`	For unicast LCK frames, the ID of the remote MEP <1-8191>
`lck_frame_type`	Type of LCK frames to send, unicast or multicast

Output Parameters

None

Return Value

Pointer to the `cfm_tst` structure when the function succeeds

cfm_throughput_rx_enable

This function is called by the `ethernet cfm throughput-measurement reception` command to enable throughput measurement on the receiving MEP.

Syntax

```
#include "cfm_tst.h"
struct cfm_tput_reception *
cfm_throughput_rx_enable (struct onmd_bridge *bridge, u_int32_t mepid,
                          u_int8_t duration, u_int8_t *md_name, u_int16_t vid)
```

Input Parameters

`bridge`	Bridge on which the MEP exists
`mepid`	ID of the source MEP <1-8191>
`md_name`	Maintenance domain name
`duration`	Length of wait time for TST frames before timing out
`vid`	VLAN ID <1-4094>

Output Parameters

None

Return Value

Pointer to the `cfm_tput_reception` structure when the function succeeds

NULL when the function fails

cfm_throughput_send_frame

This function is called by the command `ethernet cfm throughput-measurement` command to send frame throughput messages.

Syntax

```
#include "cfm_tst.h"
s_int8_t
cfm_throughput_send_frame (struct onmd_bridge *bridge, u_int32_t mepid,
                           u_int32_t rmepid, u_int8_t *md_name, u_int16_t vid)
```

Input Parameters

`bridge`	Bridge
`mepid`	ID of the source MEP <1-8191>
`rmepid`	Remote MEP ID; required for unicast frames <1-8191>
`md_name`	Name of the MD
`vid`	VLAN ID <1-4094>

Return Value

CFM_SUCCESS when the function succeeds

CFM_FAILURE when the function succeeds

cfm_vsm_send

This function is called by the `ethernet cfm vsm` command to send vendor-specific (VSM) OAM frames.

Syntax

```
#include "cfm_vsm.h"
s_int32_t
cfm_vsm_send (struct onmd_bridge *bridge, u_int8_t *md_name, u_int16_t vid,
              u_int32_t mepid, u_int32_t rmepid)
```

Input Parameters

`bridge`	Bridge on which the source MEP resides
`md_name`	Maintenance domain name
`vid`	VLAN ID <1-4094>
`mepid`	ID of the source MEP <1-8191>
`rmepid`	ID of the remote MEP <1-8191>

Output Parameters

None

Return Value

CFM_BRIDGE_NOT_FOUND when the bridge is not found

when the MD list is not found

CFM_DOMAIN_NOT_FOUND when the MD is not found

CFM_MA_NOT_FOUND when the MA is not found

CFM_MEP_NOT_FOUND when the MEP is not found

CFM_API_MEP_NOT_ACTIVE when the MEP is not active

CFM_MEMORY_NOT_ALLOCATED when memory allocation fails

CFM_RMEP_NOT_FOUND when the RMEP is not found

SNMP API

The SNMP attributes specified by the 802.1ag CFM MIB (ieee8021CfmMib) standard are supported.

Object Type	Syntax	Access	cfm_snmp_mep_config_attributes Attribute Identifier
dot1agCfmMepCcmLtmPriority	Gauge	read-write	None
dot1agCfmMepLowPrDef	Dot1agCfmLowestAlarmPri	read-write	None
dot1agCfmMepFngAlarmTime	TimeInterval	read-write	CFM_MEP_ATTRIBUTE_FNG_ALARM_TIME
dot1agCfmMepFngResetTime	TimeInterval	read-write	CFM_MEP_ATTRIBUTE_FNG_RESET_TIME
dot1agCfmMepTransmitLbmStatus	TruthValue	read-write	None
dot1agCfmMepTransmitLbmDestMacAddress	MacAddress	read-write	CFM_MEP_ATTRIBUTE_TX_LBM_DEST_MAC
dot1agCfmMepTransmitLbmDestMepId	Dot1agCfmMepIdOrZero	read-write	CFM_MEP_ATTRIBUTE_TX_LBM_DEST_MEPID

Object Type	Syntax	Access	cfm_snmp_mep_config_attributes Attribute Identifier
dot1agCfmMepTransmitLbmDestIsMepId	TruthValue	read-write	CFM_MEP_ATTRIBUTE_TX_LBM_DEST_IS_MEPID
dot1agCfmMepTransmitLbmMessages	INTEGER	read-write	CFM_MEP_ATTRIBUTE_TX_LBM_MESSAGE_COUNT
dot1agCfmMepTransmitLbmDataTlv	OCTET STRING	read-write	None
dot1agCfmMepTransmitLbmVlanPriority	INTEGER	read-write	None
dot1agCfmMepTransmitLbmVlanDropEnable	TruthValue	read-write	None
dot1agCfmMepTransmitLtmFlags	OCTET STRING	read-write	None
dot1agCfmMepTransmitLtmTargetMacAddress	MacAddress	read-write	None
dot1agCfmMepTransmitLtmTargetMepId	Gauge	read-write	CFM_MEP_ATTRIBUTE_TX_LTM_TARGET_MEPID
dot1agCfmMepTransmitLtmTargetIsMepId	TruthValue	read-write	CFM_MEP_ATTRIBUTE_TX_LTM_TARGET_IS_MEPID
dot1agCfmMepTransmitLtmTtl	Gauge	read-write	CFM_MEP_ATTRIBUTE_TX_LTM_TTL
dot1agCfmMepTransmitLtmEgressIdentifier	OCTET STRING	read-write	None

MIB Functions

This section describes functions related to reading and writing MIBs.

Function	Description
cfm_snmp_mep_config_attributes	Sets a MEP attribute
var_dot1agCfmMepTable	Reads an SNMP variable in the ieee8021CfmMib table
write_dot1agCfmMepTable	Sets an SNMP variable in the ieee8021CfmMib table

cfm_snmp_mep_config_attributes

This function sets a MEP attribute.

Syntax

```
#include "onmd/cfm/cfm_api.h"
s_int32_t
cfm_snmp_mep_config_attributes (struct onmd_bridge *bridge, u_int16_t mep_id,
```

```
                                u_int8_t *md_name, u_int16_t vid,
                                u_int32_t attribute, u_int32_t val, u_int8_t *info)
```

Input Parameters

`bridge`	Bridge
`mep_id`	MEP ID for the MEP to be added
`md_name`	MD name
`vid`	VLAN ID <1-4094>
`attribute`	One of these constants from `onmd/cfm/cfmd.h`:
	`CFM_MEP_ATTRIBUTE_ACTIVE`
	`CFM_MEP_ATTRIBUTE_CCM_ENABLE`
	`CFM_MEP_ATTRIBUTE_CCM_LTM_PRIORITY`
	`CFM_MEP_ATTRIBUTE_LOW_ALARM_PRIORITY`
	`CFM_MEP_ATTRIBUTE_FNG_ALARM_TIME`
	`CFM_MEP_ATTRIBUTE_FNG_RESET_TIME`
	`CFM_MEP_ATTRIBUTE_TX_LBM_STATUS`
	`CFM_MEP_ATTRIBUTE_TX_LBM_DEST_MAC`
	`CFM_MEP_ATTRIBUTE_TX_LBM_DEST_MEPID`
	`CFM_MEP_ATTRIBUTE_TX_LBM_DEST_IS_MEPID`
	`CFM_MEP_ATTRIBUTE_TX_LBM_MESSAGE_COUNT`
	`CFM_MEP_ATTRIBUTE_TX_LBM_DATA_TLV`
	`CFM_MEP_ATTRIBUTE_TX_LBM_VLAN_PRIORITY`
	`CFM_MEP_ATTRIBUTE_TX_LBM_VLAN_DROP_ENABLE`
	`CFM_MEP_ATTRIBUTE_TX_LTM_FLAGS`
	`CFM_MEP_ATTRIBUTE_TX_LTM_TARGET_MAC`
	`CFM_MEP_ATTRIBUTE_TX_LTM_TARGET_MEPID`
	`CFM_MEP_ATTRIBUTE_TX_LTM_TARGET_IS_MEPID`
	`CFM_MEP_ATTRIBUTE_TX_LTM_TTL`
	`CFM_MEP_ATTRIBUTE_TX_LTM_EGRESS_ID`
	`CFM_MEP_ATTRIBUTE_TX_LBM_TIME_OUT_VALUE`
`val`	Value of numeric attribute
`info`	Value of string attribute

Output Parameters

None

Return Values

CFM_BRIDGE_NOT_FOUND when the bridge is not found

CFM_API_ERR_MD_NOT_FOUND when the MD name is not found

CFM_API_ERR_INTERNAL when the MD list is NULL or the loopback for the MEP is NULL

CFM_API_ERR_MA_NOT_FOUND when the MA is not found

CFM_API_ERR_MEP_NOT_FOUND when the MEP is not found

CFM_API_ERR_INVALID_LBM_COUNT when the number of loopback messages is invalid

CFM_SUCCESS when the function succeeds

var_dot1agCfmMepTable

This function is called when an external network management system queries an SNMP variable in the ieee8021CfmMib table.

Syntax

```
#include "onmd/cfm/cfm_ieee8021CfmMib.h"
unsigned char  *
var_dot1agCfmMepTable(struct variable *vp, oid * name,
                      size_t *length, int exact,
                      size_t *var_len, WriteMethod ** write_method,
                      u_int32_t vr_id)
```

Input Parameters

`vp`	Pointer to SNMP variable structure
`name`	Pointer to the variable name (OID)
`length`	Number of elements (sub-ids) in the name
`exact`	Whether this request is a GET (exact match: PAL_TRUE) or a GETNEXT (PAL_FALSE)
`vr_id`	Unused

Output Parameters

`var_len`	Length of the variable that was read
`write_method`	Pointer to a pointer to the SET function (WriteMethod) for the variable

Return Value

A pointer to the value of the variable when the function succeeds

Zero (0) or NULL when the function fails

write_dot1agCfmMepTable

This function is called when an external network management system sets an SNMP variable in the ieee8021CfmMib table.

Syntax

```
#include "onmd/cfm/cfm_ieee8021CfmMib.h"
int
write_dot1agCfmMepTable(int action, u_int8_t * var_val,
                        u_int8_t var_val_type, size_t var_val_len,
                        u_int8_t * statP, oid * name,
                        size_t name_len, struct variable *vp,
                        u_int32_t vr_id)
```

Input Parameters

`action`	Unused
`var_val`	Value of the variable
`var_val_type`	Data type of the variable; this constant from lib/asn1.h which corresponds to an ASN.1 built-in simple type
`var_val_len`	Length of the variable
`statP`	Unused
`name`	Pointer to variable name (OID)
`length`	Length of the name
`vp`	Pointer to SNMP variable structure
`vr_id`	Unused

Output Parameters

None

Return Values

SNMP_ERR_NOERROR when the function succeeds

SNMP_ERR_GENERR when the function fails

SNMP_ERR_NOACCESS when the row status is active

SNMP_ERR_WRONGVALUE when `var_val` is invalid

SNMP_ERR_BADVALUE when `var_val` is not a valid MAC address

SNMP_ERR_NOCREATION when the MEP does not exist

CHAPTER 3 Provider Bridging

Provider bridging is defined in the IEEE 802.1ad standard which introduces a tag called the Service VLAN (S-VLAN) tag that addresses the issue that a 12-bit VID space is insufficient for addressing public services. Provider Bridging enables providers to supply Ethernet Virtual Circuit (EVC) services to their customers.

Overview

An EVC is recognized by an S-VLAN, and the packets through the provider network are doubly tagged.:

- The inner tag is called the C-VLAN tag (Customer VLAN ID)
- The outer tag is called the S-VLAN tag (Service VLAN) for the provider network.

The Enhanced Internal Sub layer Services (EISS) recognizes both service and customer VLAN tags. In the Rapid Spanning Tree Protocol (RSTP) edge bridges in the provider network participate in customer spanning tree calculations to avoid loops due to customer device misconfigurations.

The IEEE 802.1ad standard enhances the bridging functionality as defined in 802.1Q by providing separate instances of 802 MAC, MAC Internal and EISS to multiple customers independently, without requiring the customers' cooperation. To achieve this objective, these concepts are introduced:

- A VLAN bridge, consisting of a single C-VLAN component
- An S-VLAN bridge, consisting of single S-VLAN component without a C-VLAN component
- A Provider Edge Bridge (PEB), a single S-VLAN component, and one or more C-VLAN components

A Provider Bridge denotes an S-VLAN bridge, a PEB, or both.

C-VLAN and S-VLAN Components

The standard defines the S-tag to do switching in the service provider domain. It differentiates between, and provides format for C-VLAN tags used by the customer in their network, and provider VLAN tags used by the provider, by assigning different Ethernet types to each tag. The existing tag has been named as the C tags. To operate on these two tags the standard defines and specifies the functionality of two bridges: The S-VLAN component switches based on the S-tag and C-VLAN component that switches based on the C-Tag. The customer-facing side operates on customer (C-VLAN) tags and BPDUs. The provider-facing side operates on service (S-VLAN) tags and BPDUs. The interconnection between a Customer bridge and Provider Bridge uses a virtual port called the Provider Edge Port (PEP). The standard specifies the customer interfaces to the Provider Bridge network in terms of the operation and configuration of the VLAN-aware bridge components of Provider Bridges, including interfaces that offer the following:

- Provide access to a single service instance through a Bridge Port (port-based service)
- Allow a customer to select amongst and identify service instances by Customer VLAN Identifier (C-VID), or C-tagged services
- Allow a customer to select amongst and identify service instances by Service VLAN Identifier (S-VID), or S-tagged services

Bridge ports are classified as Customer Network Port (CNP), Provider Network Port (PNP), or a Customer Edge Port (CE port). A PNP transmits and receives frames for multiple customers. A CNP provides either S-tagged or port-based services to a single customer. A CE port provides a C-tagged interface to a single customer. The relay entity switches between a customer edge port or a customer network port, and provider network ports. Data is not switched between customer ports.

Logic Flows for Major Events

The diagrams in this section describe the logic flows that are followed for major Provider Bridge events.

Adding or Deleting a Bridge or VLAN

The following figure displays the flow followed when a bridge or VLAN is added or removed, a port type is set, or when adding or deleting a C-VLAN or and S VLAN from a port. Each module updates the database at the time of the event.

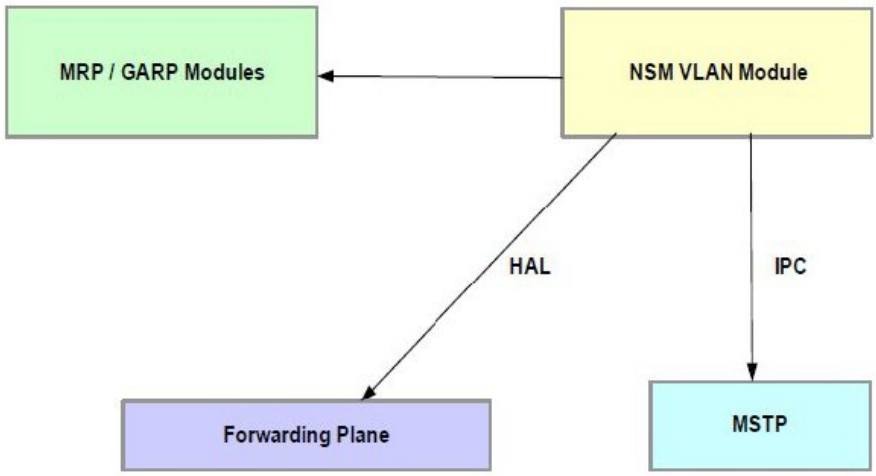


Figure 3-1: Adding or Deleting a Bridge or VLAN

Configuring a Port as Customer Edge Port

The following figure displays the flow followed when a port is configured a Customer Edge (CE) port.

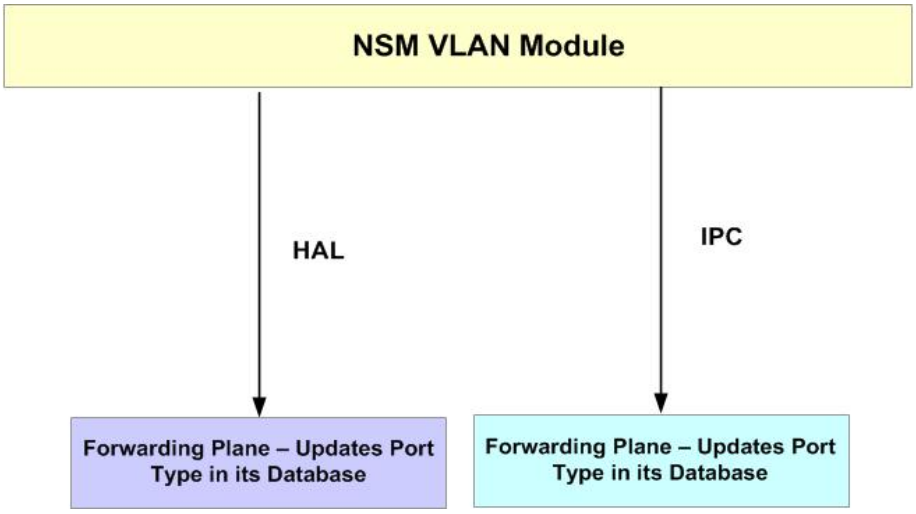


Figure 3-2: Configuring a Port as CE Port

Configuring a C-VLAN to S-VLAN Mapping for CE Port

The following figure displays the flow followed when a C-VLAN to S-VLAN mapping is provided for port that is configured as a CE port.

1. NSM VLAN module sends notification to MSTP only when a new S-VLAN mapping is added, which results in the addition of a PEP. The forwarding plane is notified of all C-VLAN to S-VLAN mapping changes.
2. Forwarding plane updates C-VLAN to S-VLAN Mappings in its database.
3. MSTP adds a PEP to the virtual bridge for the corresponding CEP.

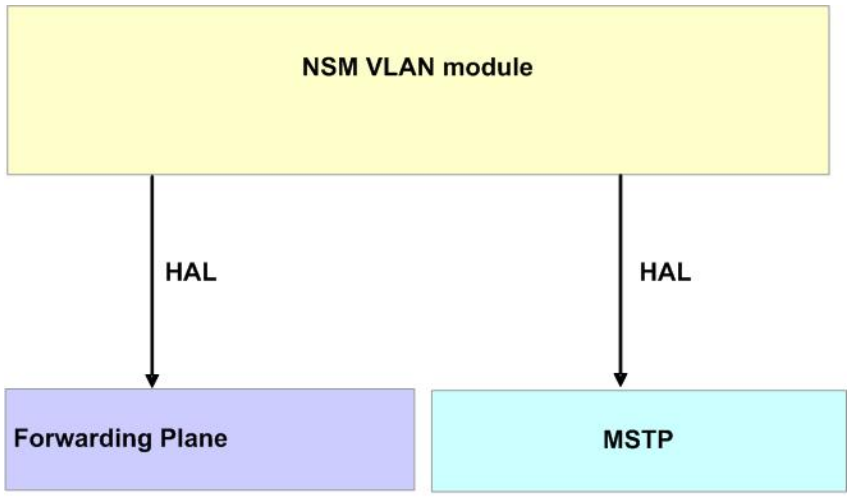


Figure 3-3: Configuring a C-VLAN to S-VLAN Mapping for CE Port

Configuring S-VLAN to C-VLAN Translation

The following figure displays the flow followed when an S-VLAN to -VLAN translation table is configured.

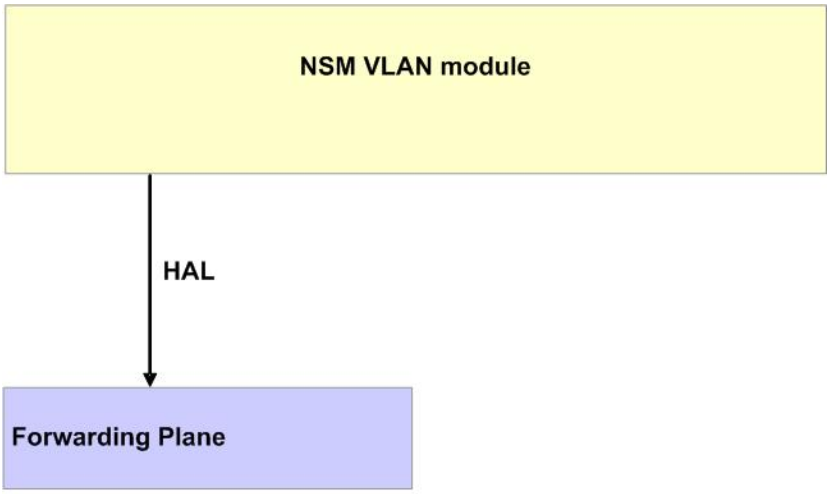


Figure 3-4: Configuring S-VLAN to C-VLAN Translation

Managed Objects

The following managed objects are supported.

- Read and configure a provider bridge port type as customer network, customer edge and provider network.
- Read and configure a VID translation table for provider network or customer network port.
- Read and configure customer edge port parameters like the C-VLAN registration table, add the customer edge port or provider edge port to the tagged or untagged set of the C-VLAN.
- Read and configure the provider edge port parameters like the P-VID, acceptable frame types, and enable or disable ingress filtering.

Note: The customer network, provider network and customer edge port configurations are done in the interface mode of the respective interfaces. The configuration of the provider edge port, which is a logical port, is done in the interface mode corresponding to the customer edge port.

CFM in a Provider Edge Bridge Network.

This section describes Connectivity Fault Management (CFM) support in a Provider Edge Bridge (PEB) network. This feature provides end-to-end service instance monitoring across Provider Bridging (PB) networks and a mechanism for CFM frames to flow across PB networks in compliance with IEEE 802.1ag 2007 Section 22.6, MEF 17, and MEF 20.

Overview

When there is a C-VLAN and an S-VLAN mapping located on a PEB, sent frames are relayed toward the PN port, where there is an S-VLAN (EVC) configured and where the frames enter the PB core (that is, bridge 3, 4, etc.). If this mapping is not present at OAM Network Management Daemon (ONMD), the relay function of CFM attempts to relay the frame towards the PB core with the received C-VID (Customer VLAN Identifier), since the C-VLAN is not configured on any of the PNPs the frame is not relayed and it is dropped. A user can create multiple maintenance C-VLANs if they are confident that the provider can carry each C-VLAN to the correct S-VLAN. The user must create one maintenance C-VLAN per provider service instance (S-VLAN), thus creating one maintenance C-VLAN per PE port in the C-VLAN.

By configuring each PE Port to pass a C-VLAN untagged frame to a corresponding Customer Network (CN) port in the S-VLAN component, the Protocol Data Unit (PDU) of the CFM in one Maintenance Association (MA) is visible to the S-VLAN component. Moreover, the provider can then configure a Maintenance Intermediate Point (MIP) for each S-VID (Service VLAN Identifier) that is visible to the MA of the user. This allows the user to have one MA protecting each service instance.

System Architecture

When the system receives a CCM frame, it first checks to determine the port type, which is either the CE port or the PN port for the PEB. If it is destined for the CE port, then the frame is passed to the PE port, where there is a configured maintenance C-VLAN. Otherwise, the user needs to configure the default VID on the PE port as maintenance C-VLAN. The modules into which this requirement divides to support system architecture include the following:

- Setting up Internal PE Port on page 93
- Support PB related information at ONMD on page 93
- End-to-end Convergence on page 93
- MIP on CN port on page 93
- Down MEP on CN port on page 95
- UP MEP on PN port on page 96

Setting up Internal PE Port

When a user configures an S-VLAN to C-VLAN mapping on the CEP, the PEP is created internally. Both the S-VLAN to C-VLAN mapping and the internal PEP are maintained at the NSM. Once created, NSM informs ONMD for the PEP and the ONMD is then created internally to maintains the PEP. This module resides on the ONMD to interact with the NSM to receive the necessary bridge and port type information. In addition, the module depends on the underlying lower layer (that is, the software forwarder) to provide the C-VID and S-VID for incoming frames on the CE port. When configured, this module helps the internal PE port to facilitate CFM frame flow from end-to-end (that is, customer equipment to customer equipment across the PB core).

The CE port of a PE bridge maintains an internal PE port. The user does not have to create internal ports for the PE port or the CN port for the provider bridge. All flow points for the CN port are configured with an S-VLAN context on the CE port.

Support PB related information at ONMD

Users must set up ONMD to support a provider bridge topology, which is specific to CFM flow points. ONMD supports message handlers for PB messages that are configured in CFM for PB-specific maintenance entities. PB information at the NSM, including port types and default VIDs of the PB ports, are received and stored at the ONMD. In addition, users can setup rules so that the ONMD can use the auto-MIP creation algorithm and create new MIPs.

End-to-end Convergence

CFM in a PEB allows end-to-end convergence of MEPs (Maintenance Endpoints) configured on customer equipment in either a subscriber or a customer domain. CFM frames originating on the customer equipment flow through the PB via the PEB. Moreover, end-to-end convergence of MEPs is seen on peer customer equipment. Tagged or untagged customer equipment CFM frames entering the CE port relay to both the internal PE port and the internal CN port and are seen on the physical PN Port. Single-tagged (S-Tag) and double-tagged (C-Tag and S-Tag) frames are seen in the PB core, based on how the user configured the PE port. MEPs configured on customer equipment are able to receive and accept CCMs from another MEP that was configured on customer equipment in the customer domain.

MIP on CN port

In compliance with MEF 17 and MEF 20, users can configure a CFM MIP in the subscriber domain on the UNI-N. There is no MIP on the CE port or on the PE port. However, there is an S-VLAN-aware (s-component) MIP on the CN port. Since the CN port is an internal port, users can configure the MIP as an S-VID-aware MIP. This allows for convergence, which is the intention of the IEEE and the MEF standards.

MIP Responds to an LBM Sent Towards the Customer Equipment

The following is the way the MIP responds to a lookback message (LBM) from the CN port that was sent towards the customer equipment:

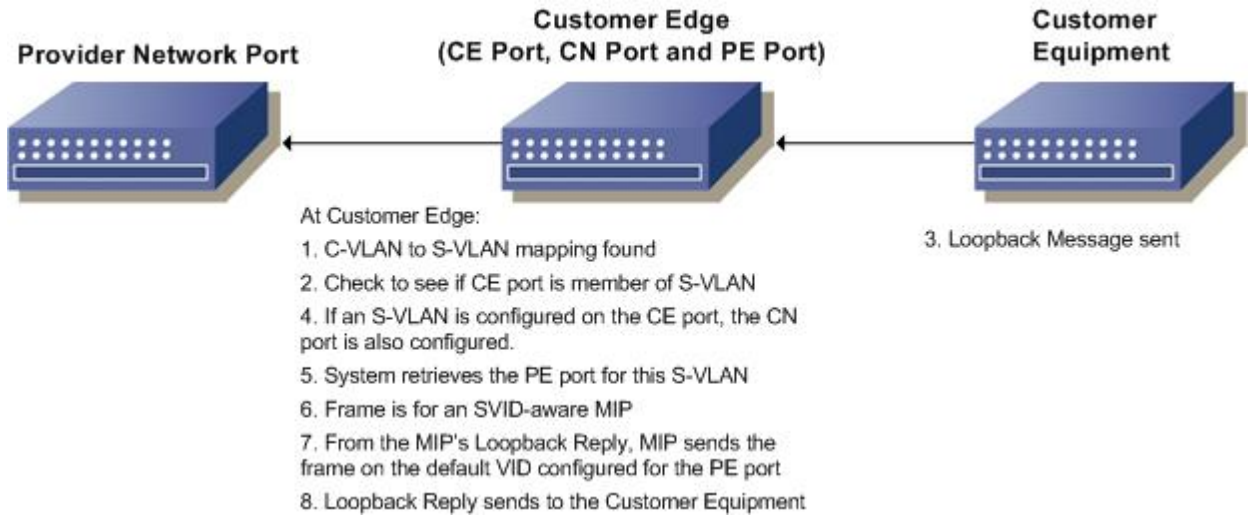


Figure 3-5: MIP Responds to an LBM Sent Towards the Customer Equipment

MIP Response for LBM Sent Towards the PB Core

The following is the MIP response to a loopback message from the CN port that was sent towards the PB core:

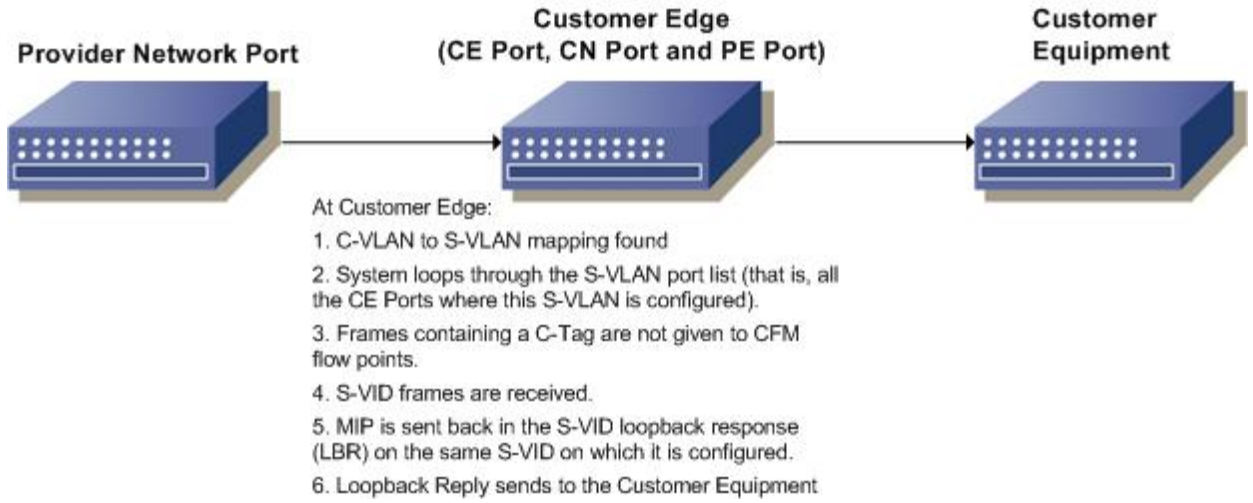


Figure 3-6: MIP Response for LBM Sent Towards the PB Core

MIP Response for LTM Sent Towards the CE

The following is the MIP response for a link trace message (LTM) from a CN port that was sent towards customer equipment:

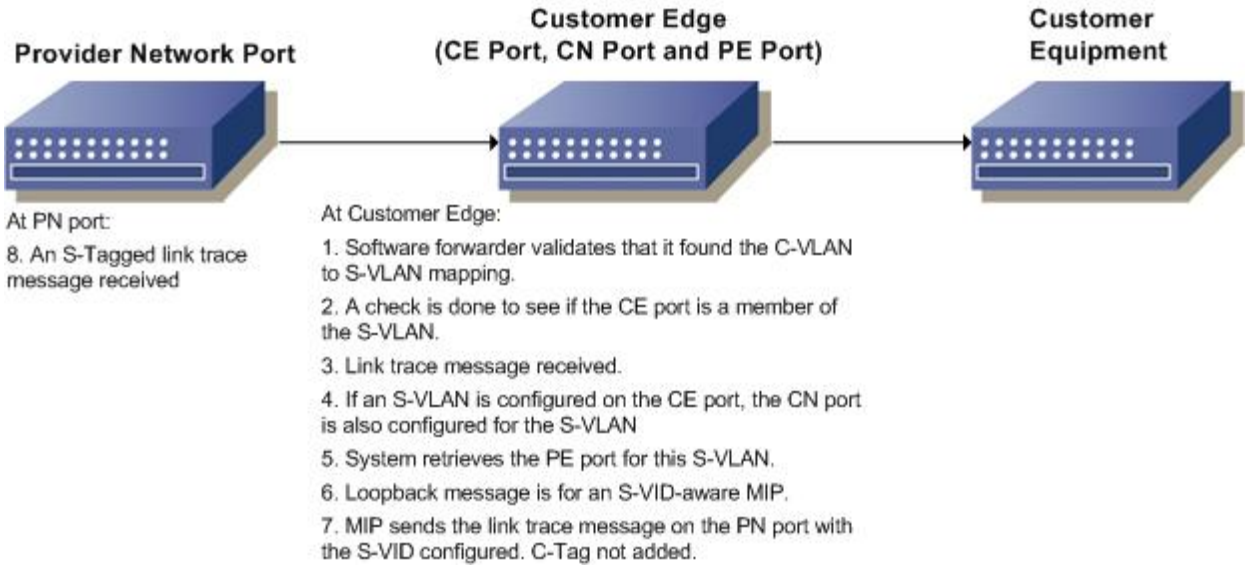


Figure 3-7: MIP Response for LTM Sent Towards the CE

MIP Response for LTM Sent Towards the PB Core

The following is the MIP response for a link trace message from the CN port that was sent towards the PB core:

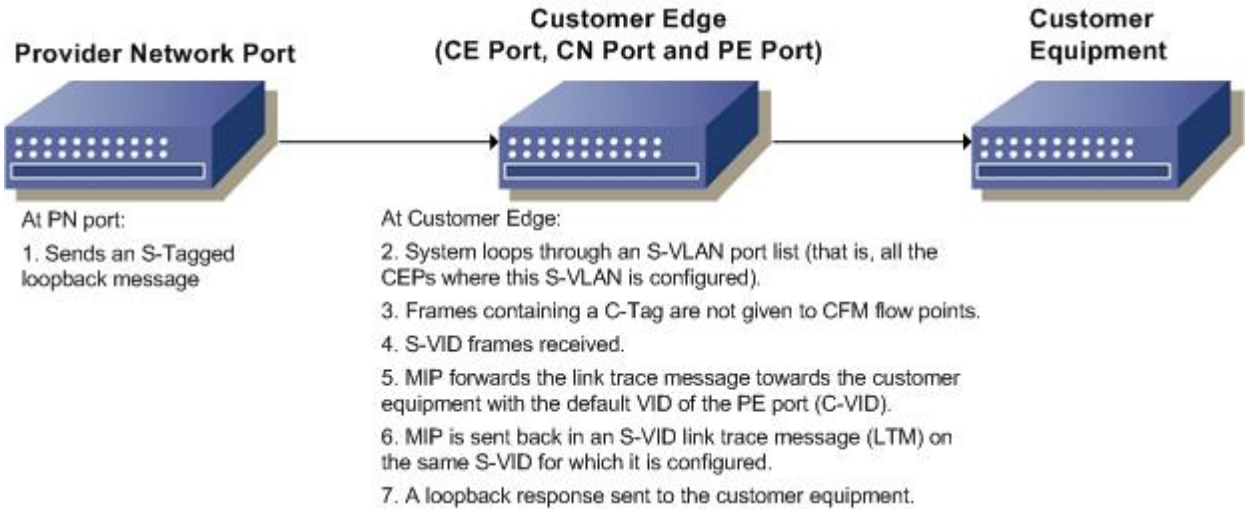


Figure 3-8: MIP Response for LTM Sent Towards the PB Core

Down MEP on CN port

If an S-VLAN-aware MEP fails on S-component, the MEP sends CFM frames in the customer domain to the mapping PE port and on the wire from the CE port. A configured MEP on an S-VLAN-aware internal CN port sends frames out of the CE port towards the customer equipment in either the customer or the subscriber domain. Users can monitor the link from the customer equipment until it reaches the PEB s-component (that is, before it reaches the core).

Response from a Down MEP on CN Port

The following is the response from a down MEP on a CN port, which was sent towards customer equipment.

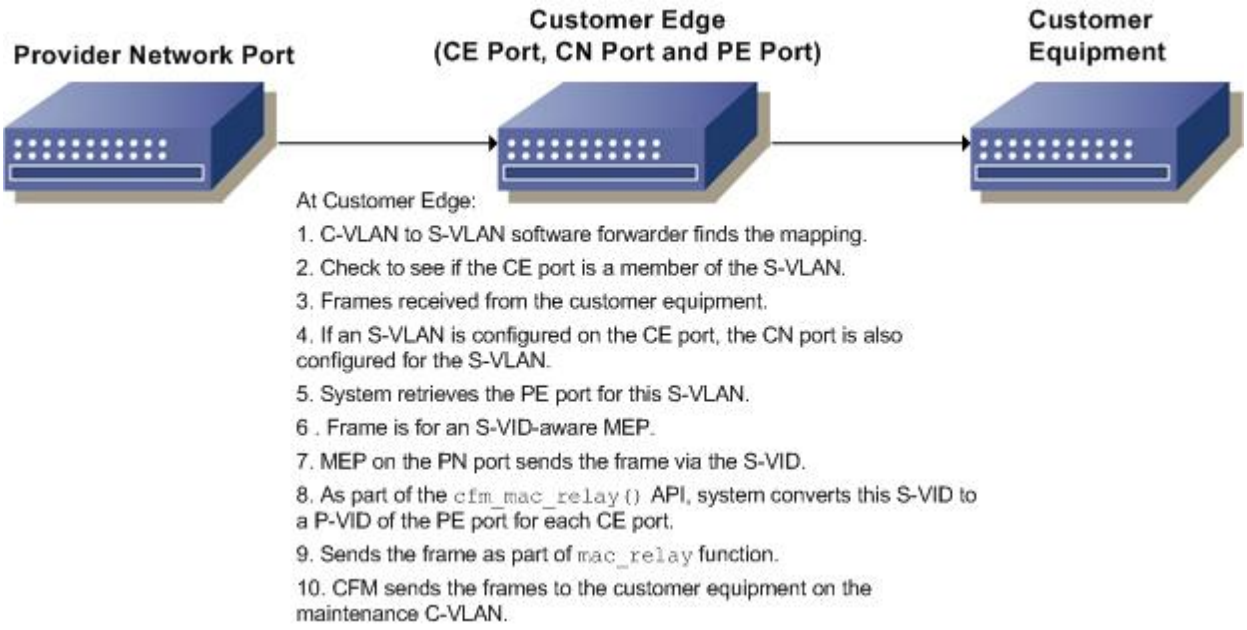


Figure 3-9: Response from a Down MEP on CN Port

UP MEP on PN port

An up MEP setup on the s-component sends CFM frames from the customer domain to the mapping CN port or PE port from the CE port. a MEP configured on an S-VLAN-aware PN port sends frames from the CE port towards the customer equipment in either the customer or the subscriber domain. Users can monitor this S-VLAN across multiple CN or PE ports.

Response from an Up MEP on PN Port

The following is the response from an up MEP on a PN port, which was sent towards customer equipment.

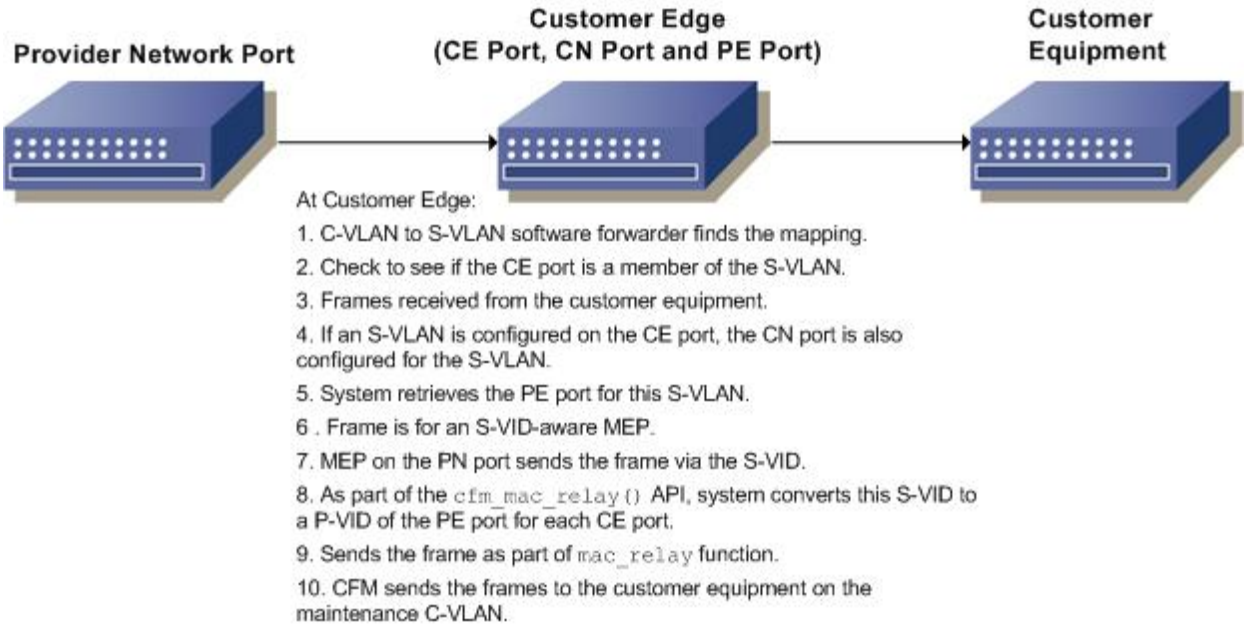


Figure 3-10: Response from an Up MEP on PN Port

Frame Flow for Tagged Frames

The following describes the frame flow for tagged frames coming into the CE port. An untagged VLAN configured on PE port is same as the C-Tag.

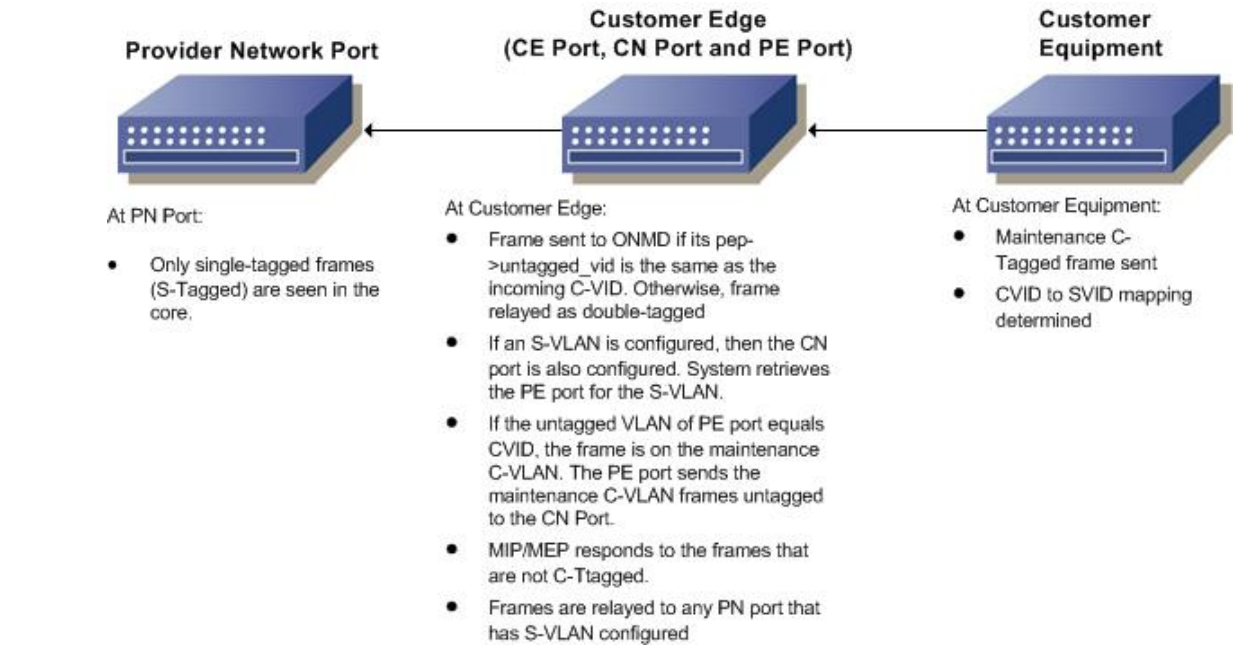


Figure 3-11: Frame Flow for Tagged Frames

Frame Flow for Frames Coming from PN port

The following describes the frame flow for tagged frames coming from PN port.

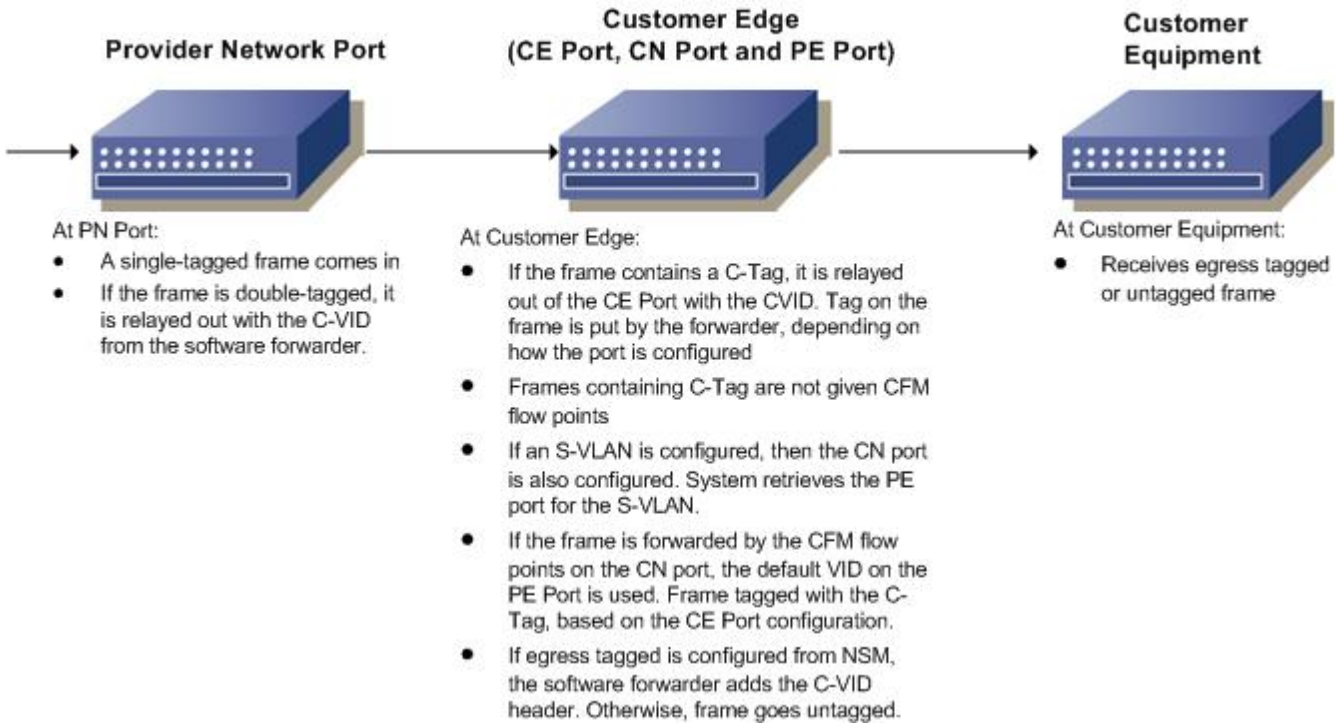


Figure 3-12: Frame Flow for Frames Coming from PN port

Interfaces

Connectivity Fault Management (CFM) support on a Provider Edge Bridge (PEB) includes the following interfaces.

NSM-ONMD

NSM sends information regarding the bridge type (as Edge Bridge) and the port type. The encoding/decoding for the `nsm_msg_bridge` API structure is also updated.

Layer2/swfwdr - ONMD

The software forwarder (swfwdr) function the ONMD the C-Tag, and the S-Tag, for the incoming frames on the CE port. The C-VID and S-VID mapping maintained at the forwarder is reused. The S-VID are filled in at the l2_skaddr>S-VLAN ID and then sent to the ONMD. On the CE port, if the C-VID equals the PE port->untagged_vid, the swfwdr double-tags the frame (that is, S-Tag and C-Tag) and relays it of the PNPs, is not sent to the ONMD. On the PN port, if the frame has a C-Tag (that is, double-tagged frame received), it does not send it to the ONMD. Based on the C-VID, it is only related to the CE ports.

For all incoming frames, priorities are decoded at the swfwdr and sent to ONMD. On the PB core bridge, the frames are relayed with the priority with which they were received. To relay a C-VLAN frame (from the CE port) on the PEB, the configured S-VLAN priority is used to relay the frame out to the PN port. If the frame is an S-VLAN frame (on the PN port), then it relays out of the CE port with the configured priority for that C-VID (PE port default VID).

System Design

Of all C-VLANs mapped to an S-VLAN, a user can configure only one C-VLAN as a maintenance C-VLAN. Moreover, users are required to configure this C-VLAN on the PE port as the default VID. When a CN port receives a frame from a PE port on this VID, the CFM frame is untagged. In addition, there can be many C-VLANs mapped to the S-VID on the PE port. However, only one can be the maintenance C-VLAN. When an untagged CFM frame comes from the CN port to the PE port, the PE port tags it with the configured C-VLAN value. Therefore, untagged frame on the PE port uses the default VID, which is the maintenance C-VLAN, and send the frames to both the CE port and to the MEP on customer equipment.

For a down MEP on a CN port

If a user creates a MEP on an S-VLAN ID, frames from the MEP go to the internal PE port, where there is a configured the maintenance C-VLAN as the default VID. Therefore, when the frame comes from the CE port, it contains the maintenance C-VLAN. Typically, a MEP in the customer equipment is configured for this C-VID. From the customer equipment to the PEB, users are able to monitor customer service instances. This allows users the ability to check the connectivity from inside the Edge Bridge, but not necessarily from end-to-end.

Logically, the MIP and the down MEP cannot co-exist. Thus, users that monitor customer equipment cannot do end-to-end monitoring or customer service instance monitoring, unless they are using two maintenance different levels. The auto MIP creation algorithm assures that the MIP and MEP are in the same level, and that monitoring of the same VID does not co-exist on the same port (see Down MEP on CN port on page 95 for more information).

Information from NSM to ONMD

- NSM_MSG_S-VLAN_ADD_CE_PORT
- PB Port type - CE port/PN port
- PE port - Default VID
- PE port - Tagged or untagged
- CE port - Tagged or untagged
- CE port - Default VID

Frame Flow from CE port to PN Port

The following is how frames flow from the CE port to the PN port (that is, CE port > PE port > CN port > PN port).

1. An untagged frame is received at CE port.
2. The software forwarder lifts the frame to the ONMD with the default VID of the CE port.
3. A comparison is made to the VID received at the ONMD with the untagged VID of the PE port.
4. If the received C-VID equals the untagged VID of the PE port, then only the S-Tagged frame is a double-tagged frame.
5. The system finds the S-VLAN for the C-VLAN on CE port.
6. For this S-VLAN, the system does a lookup on the MIP/MEP of the CE port for non-C-Tagged frames (this is because MIP/MEP acts upon only S-VLAN).
7. If the received C-VID equals the untagged VID on the PE port, then this is a double-tag frame.
8. For double-tagged build C-Tag at ONMD, the frame is sent out of the PN port with the S-VLAN configured.
9. The software forwarder adds an S-Tag to the frame.

Frame Flow from PN port to CE Port

The following is how frames flow from the PN port to the CE port (that is, PN port > CN port > PE port > CE port)

1. The system picks up the S-VID frame from PN port.
2. The S-VLAN finds the CE port. The CE port must have S-VLAN membership.
3. After finding the CE port, any non-C-Tagged frames looks for the MIP/MEP of this S-VLAN.
4. In addition, for this S-VLAN, there is a PE port that sends the frame on the default VID of the PE port (the default VID of PN port is configured from NSM).
5. If the frame has a C-Tag, then the lookup for MIP/ME does not take place at the CE port for the S-VLAN.
6. If egress tagging is configured, the frame is sent from the CE port, and the software forwarder adds the egress tag.

Data Flow Description

The following section describes the different logical frame flow for Connectivity Fault Management (CFM) support on a Provider Edge Bridge (PEB).

CE Port Receives Untagged Frames

The following frame flow is when the CE port receives an untagged frame with the PE port configured with an untagged C-VID.

1. The customer equipment sends an untagged frame.
2. If the default VID of the CE port equals the PE port's untagged VID, then the frame is sent to the ONMD. Otherwise, it is relayed from software forwarder.
3. At the ONMD (CE port), a check is done to see if the CE port is a member of the S-VLAN.
4. If an S-VLAN is configured on the CE port, then the CN port is also configured for the S-VLAN. Therefore, the system retrieves the PE port for this S-VLAN.

5. If the untagged VLAN ID equals the VID in a frame, then this frame is not double-tagged. This means that the PE port is configured to send untagged frames to CN port.
6. The MIP/MEP only responds to frames that are not C-Tagged.
7. From the CE port, the frames are relayed to all PN ports that have S-VLAN configured.
8. Only single-tagged frames (S-Tagged) are seen in the core.

CE Port Receives Untagged Frames

The following frame flow is when the CE port receives an tagged frame with the PE port configured with different untagged C-VID (maintenance C-VID).

1. The customer equipment sends a tagged frame.
2. If the C-VID equals the PE port > untagged_vid, then the frame is sent to the ONMD. Otherwise, the frame is relayed double-tagged.
3. At the ONMD (CE port), a check is done to see if the CE port is a member of the S-VLAN.
4. If an S-VLAN is configured on the CE port, then the CN port is also configured for the S-VLAN. Therefore, the system retrieves the PE port for this S-VLAN.
5. If the untagged VLAN ID equals the VID in a frame, then the frame is not double-tagged. This means that the PE port is configured to send untagged frames to CN port.
6. The ONMD or software forwarder builds the C-Tag in the frame.
7. Since this is a C-tagged frame, the system does not check for MEP/MIPs in the S-VLAN mapping to this C-VID. The frame is simply relayed.
8. From the CE port, the frames are relayed to all PN ports that have S-VLAN configured.
9. Double-tagged frames are seen in the core. The software forwarder adds S-Tag.

Frame Flow for Frames Coming from the PN Port

The following frame flow is for frames coming from the PN port

1. The PN port sends an S-Tagged frame.
2. The system loops through the S-VLAN port list (that is, all CE ports where this S-VLAN is configured).
3. Once the CE port is identified, the system checks whether the frame contains a C-Tag.
4. If the frame contains a C-Tag, it is relayed from the CE port with that C-VID. The software forwarder puts a tag in the frame, based on how the port is configured.
5. Frames containing a C-Tag are not given to CFM flow points.
6. If the frame has to be forwarded by CFM flow points (LTM) on the CN port, then the default VID of the PE port is used (C-VID). The frame is tagged with a C-Tag based on the configuration of the CE port.
7. If egress tagging is configured for the frame in NSM, then the software forwarder adds the C-VID header. Otherwise, the frame goes untagged.
8. The egress-tagged or untagged frame is sent to the customer equipment.

Frame Flow for Double-tagged Frames Coming from the PN Port

The following frame flow is for double-tagged frames coming from PN port:

1. The PN port sends a double-tagged frame.
2. The S-Tag is stripped and the frame is put on CE port with the C-Tag.
3. An MIP/MEP check is note done, since the frame has a C-Tag.
4. CE port sends the frame with the C-VID that came with the frame.
5. If egress tagging is configured for the frame in NSM, then the software forwarder adds the C-VID header. Otherwise, the frame goes untagged.
6. The egress-tagged or untagged frame is sent to the customer equipment.

Frame Flow for Double-tagged Frames Coming from the PN Port

The following frame flow is for double-tagged frames coming into a CE port, with an untagged VLAN configured on PE port that is the same as the C-Tag.

1. CE sends a maintenance C-Tagged frame.
2. The frame is sent on the CE port with the incoming C-VID.
3. At the CE port, the C-VLAN mapping is found.
4. A check is done to see if the CE port is a member of the S-VLAN.
5. If an S-VLAN is configured on the CE port, then the CN port is also configured for the S-VLAN. Therefore, the system retrieves the PE port for this S-VLAN.
6. MIP/MEP responds to only the frames that are not C-Tagged.
7. From the CE port, the frames are relayed to all PN ports that have S-VLAN configured.
8. Only single-tagged frames (S-Tagged) are seen in the core.
9. A MIP/MEP check is not done, since the frame has a C-Tag.

CHAPTER 4 Provider Backbone Bridging

The ZebOS-XP Provider Backbone Bridging (PBB) implementation is based on the IEEE 802.1ah standard. This specification addresses the issue that the 12-bit S-VID Provider Bridging (as defined in IEEE 802.1ad) service instance space is insufficient for addressing public services. To alleviate this limitation, the draft specification introduces a tagged MAC header (B-MAC and B-tag), and a service instance tag (I-tag).

Overview

Provider Backbone Bridging enables a provider organization to build an 802.1ah-compliant Ethernet backbone network of multiple, inter-connecting 802.1ad Provider Bridging aggregate networks. Customer service frames (S-tagged or C-tagged) are encapsulated by an outer MAC header that consists of an independent MAC address space, a B-VLAN tag, and a service instance I-tag. The Enhanced Internal Sublayer Service (EISS) recognizes these frame formats. In the multiple spanning tree protocol (MSTP), customer edge (CE) bridges support loop-free redundant connections to a PBB network.

I- and B-Component Network Edges

IEEE 802.1ah specifies two types of network edge components, an I-component and a B-component. A Backbone Edge Bridge (BEB) can be an I-BEB, a B-BEB, or an IB-BEB (a backbone edge bridge with both types of components resident). There can be multiple I-components, but only one B-component per BEB. A B-component handles B-VLAN bridging, and is Backbone Customer Bridge (BCB) facing. An I-component handles customer VLAN (C-VLAN, S-VLAN) bridging, and is customer facing. The interface between the I- and B-components is called an I-tagged interface. These two components exchange frames identified by a service instance identifier (I-SID) only.

Service Frames

In real-world situations, it is expected that multiple I-BEB (each sourcing I-tag encapsulated service frames) can connect to a single B-BEB, which maps these service frames to B-VLAN network-level transport as shown in the flow diagram below. The diagram depicts the I-BEB to B-BEB interconnection.

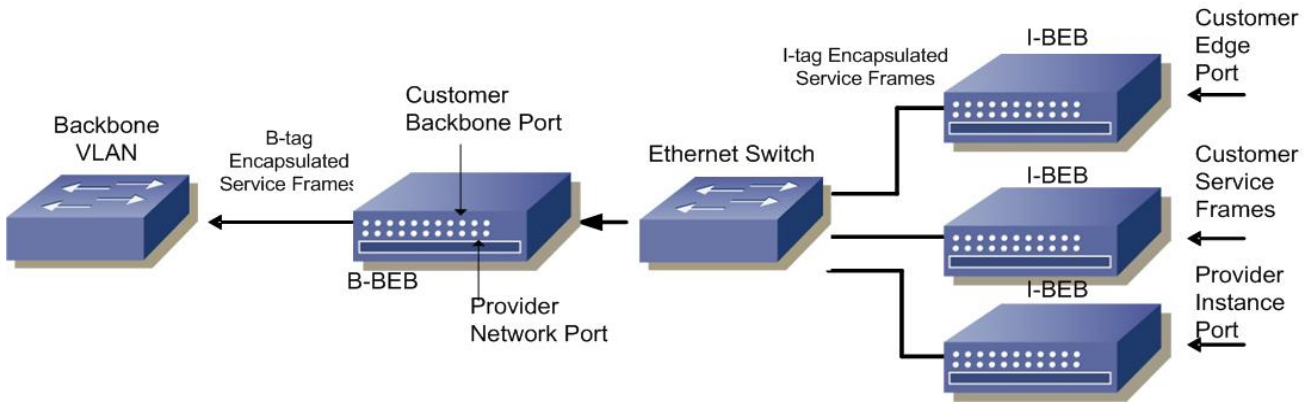


Figure 4-1: I-BEB to B-BEB Interconnection

With respect to an 802.1ad Provider Edge Bridge (PEB), it is beneficial to compare the edge-bridge architecture between Provider Bridging and Provider Backbone Bridging. Figure 4-2 depicts a Provider Edge Bridge Architecture:

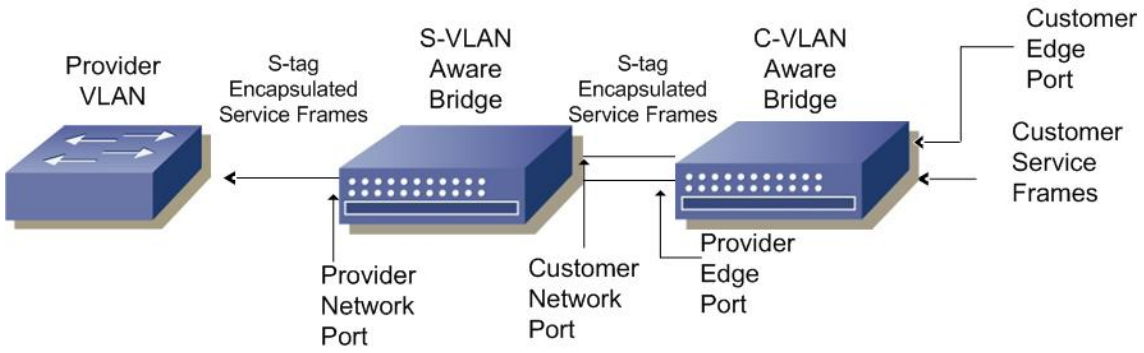


Figure 4-2: S-VLAN to C-VLAN

As shown in the two figures, Provider Edge Bridges (PEB) and Backbone Edge Bridges (BEB) use similar separated tag domain architectures to handle the additional encapsulation.

I-Component

As defined in IEEE 802.1ah, a BEB may contain zero or many I-components, and zero or one B-component. In addition, both I- and B-components are S-VLAN (service VLAN) components, and therefore are bridges. An I-component is the demarcation point between customer and backbone MAC address space. This means that the I-component must learn and maintain mappings between C-DA and B-DA (the MAC address of a destination BEB within a provider backbone bridge network (PBBN). Logically, this is what is taking place:

1. The Customer Backbone Port (CPB) receives backbone frames that contain long I-tags, which include the C-DA and C-SA elements.
2. The Provider Instance Port (PIP) creates the required C-MAC to B-MAC mappings by pairing and recording the B-SA and C-SA tuple of the backbone frame received.
3. The mapping is then employed in the reverse (ingress) direction to select a B-DA to be used by the ISS on the left side of the I-component as shown in the first figure of this chapter.
4. The PIP is the reference point where B-MAC addresses are inserted or stripped from transit service frames.

Provider Backbone Bridging

Each 802.1ah I-component is modeled in ZebOS-XP as an `nsm_bridge` bridge instance `<1-32>`. A single B-component is modeled as a separate bridge instance independent from the `<1-32>` I-component bridge instances. A pointer in the `nsm_bridge_master` structure points to Provider Backbone Bridge service-instance-specific tables (`nsm_beb_bridge`) to support PBB service provisioning. All bridge instances can be referenced from the `nsm_bridge_master`. Each `nsm_bridge{}` uses the master back-pointer to get to this pointer in `nsm_bridge_master` for accessing the PBB service instance tables.

The `nsm_beb_bridge` presents a single bridge, externally, to the network management system, while internally, it is modeled by multiple `nsm_bridge` instances that all reference the same set of `nsm_beb_bridge` component tables to collectively present the external bridge view. In other words, `nsm_bridge id` is used as the component ID to reference rows in the tables. S-tagged (one-to-one, one-to-many, one-to-all, and transparent) service interfaces, I-tagged service interfaces, and port-based PBB service interfaces are associated with ports within an `nsm_bridge` instance. The ZebOS-XP design follows the guideline that Bridge, VLAN, and bridge port-type provisioning use single-step configuration models so that changes are sent to the Hardware Service Layer (HSL) immediately, while service instance related provisioning uses a two-step, configure and commit-or-dispatch configuration model, wherein changes are sent to the HSL only when service is dispatched.

Adding or Deleting Bridge VLAN

The following figure displays the flow followed when a Bridge and VLAN is added to or removed from a port, the port type is set, or adding or deleting an S-VLAN or B-VLAN to or from a port. Each module triggers an update to the database at the time of the event.

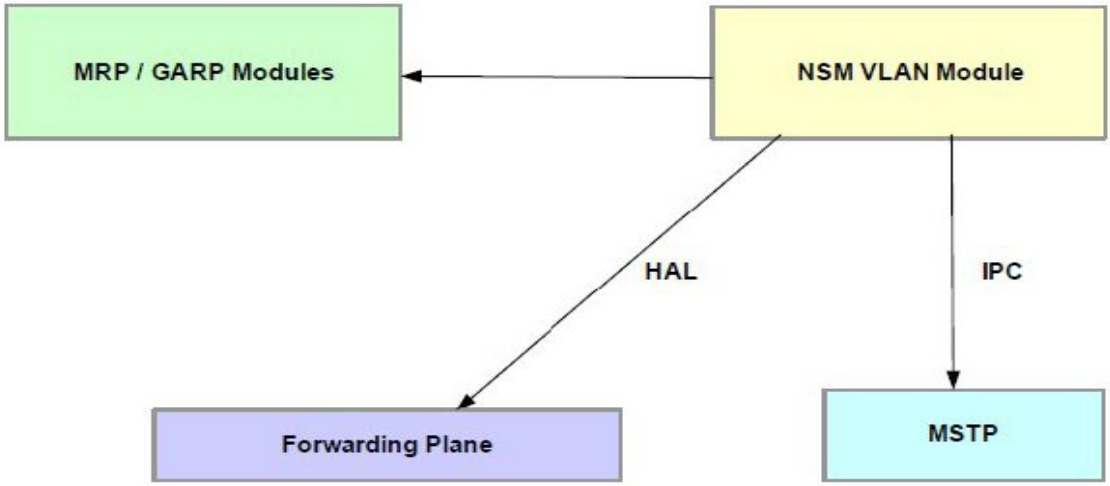


Figure 4-3: Adding or Deleting Bridge VLAN

Note: An I-BEB bridge may have only S-VLAN segments and one or more B-VLAN Bridges; a B-BEB bridge may only have B-VLAN segment and a single B-VLAN bridge; an IB-BEB may have both S-VLANs/Bridge(s) and B-VLANs/Bridge.

Bridge and VLAN-level provisioning applies hardware changes immediately, meaning that HAL messages are sent to the Forwarding Plane as shown.

Configure a Port as a Customer Network Port

The following figure displays the flow that is followed when a port is configured as a Customer Network Port (CNP).

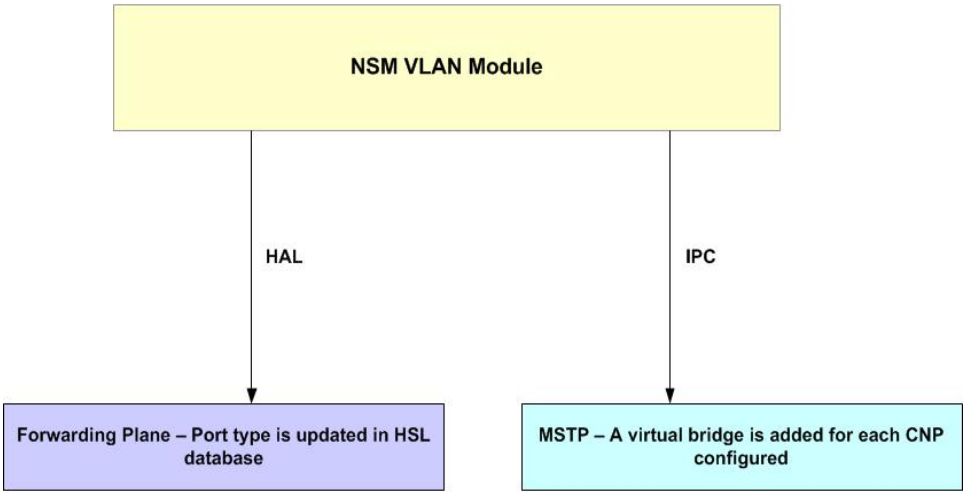


Figure 4-4: Configure a Port as a Customer Network Port

As previously described, bridge port type configuration is executed immediately by sending a HAL message to the HSL. The `nsm_beb_bridge` port list table is updated.

Configure S-VLAN to I-SID Mapping for CNP

The following figure displays the flow followed when an S-VLAN to I-SID mapping is provided for ports configured as customer network ports.

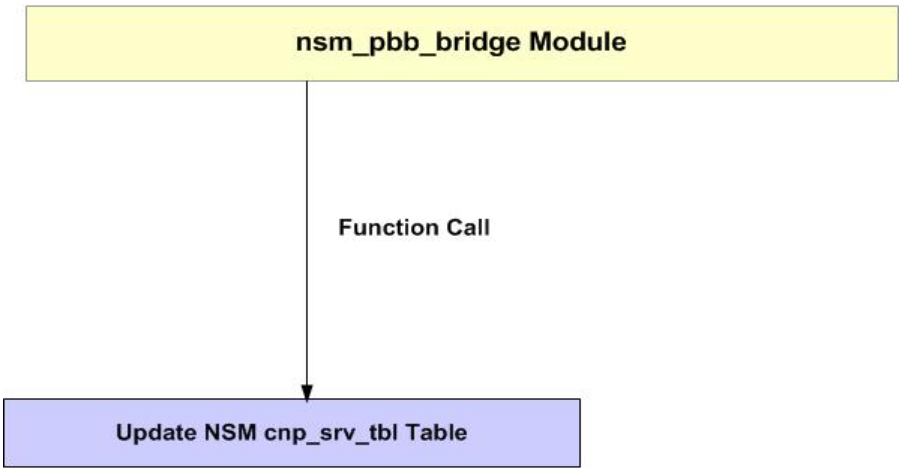


Figure 4-5: Configure S-VLAN to I-SID Mapping for CNP

Since this is service instance-related provisioning, it only requires an NSM internal table update.

Configure VIP to CIP Port Association

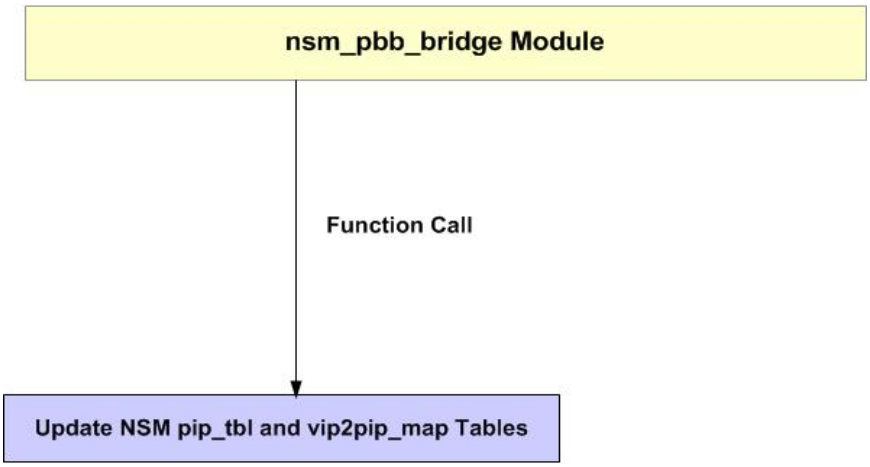


Figure 4-6: Configure VIP to CIP Port Association

Since this is service instance-related provisioning, it only requires an NSM internal table update.

Configure Port as Customer Backbone Port

The following figure displays the flow followed when a port is configured a Customer Backbone Port (CBP).

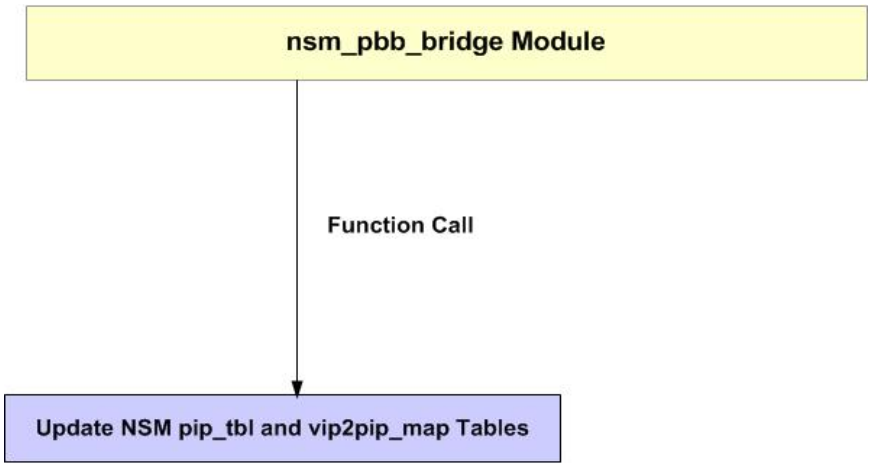


Figure 4-7: Configure Port as Customer Backbone Port

Since this is service instance-related provisioning, it only requires an NSM internal table update. The `nsm_beb_bridge` port list table is updated.

Configure I-SID to B-VID Mapping for CBP

The following figure displays the flow followed when an I-SID to B-VID mapping is provisioned for ports configured as customer backbone ports.

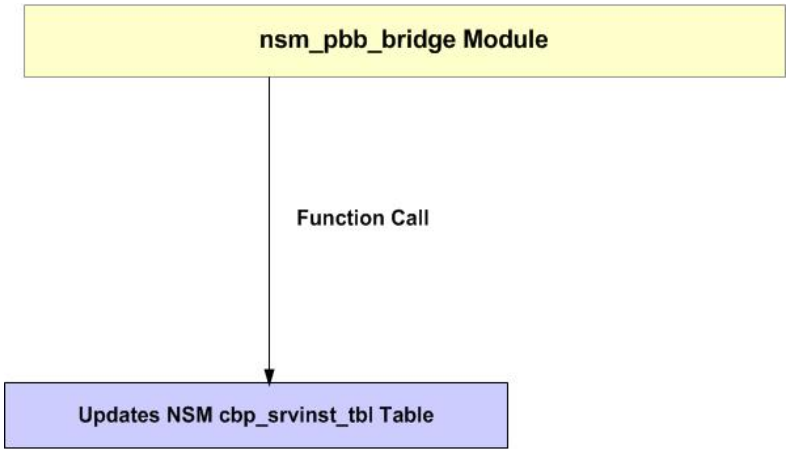


Figure 4-8: Configure I-SID to B-VID Mapping for CBP

Since this is not a service instance-related configuration, changes are dispatched to the forwarding plane immediately.

Configure Port as Provider Network Port

The following figure displays the flow followed when a port is provisioned as a Provider Network Port (PNP).

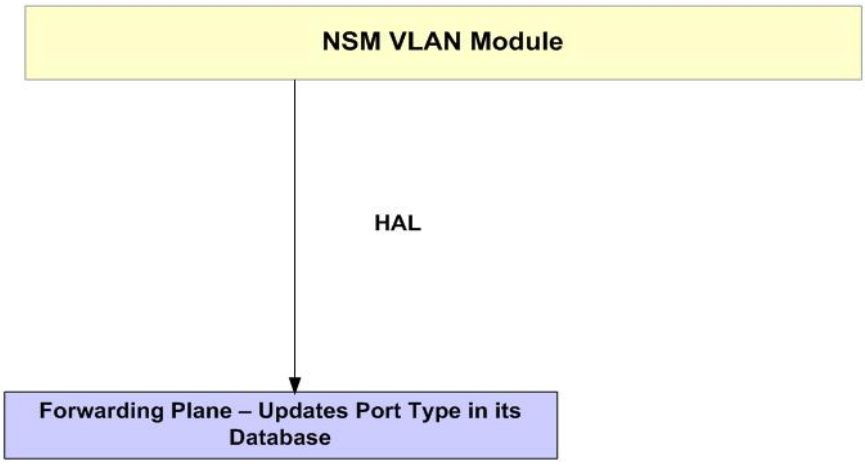


Figure 4-9: Configure Port as Provider Network Port

Dispatch Service Instance at Customer Network Port

The following figure displays the flow followed when a service provisioned at a customer network port is dispatched to the forwarding plane.

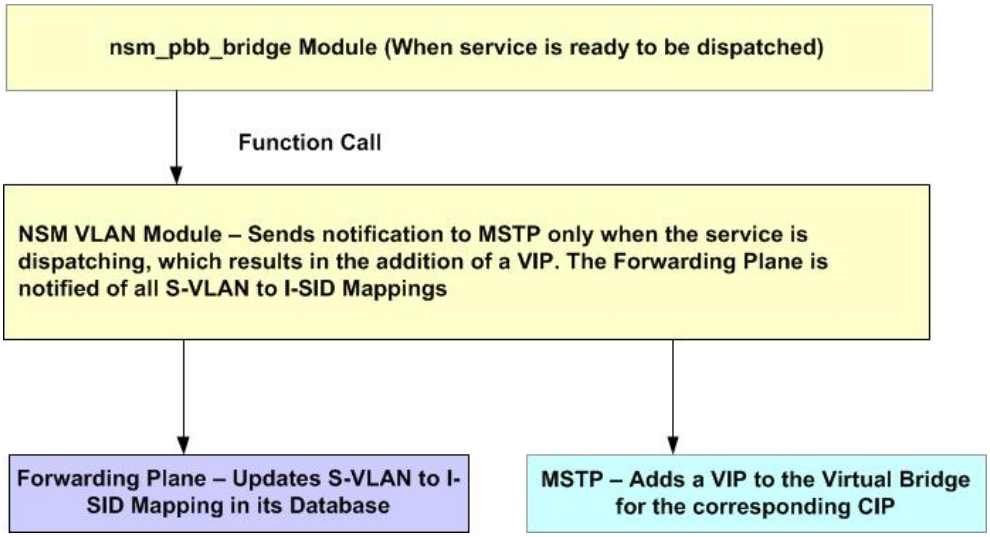


Figure 4-10: Dispatch Service Instance at Customer Network Port

Dispatch Service Instance at Customer Backbone Port

The following figure displays the flow followed when a service provisioned at a customer backbone port is dispatched to the forwarding plane.

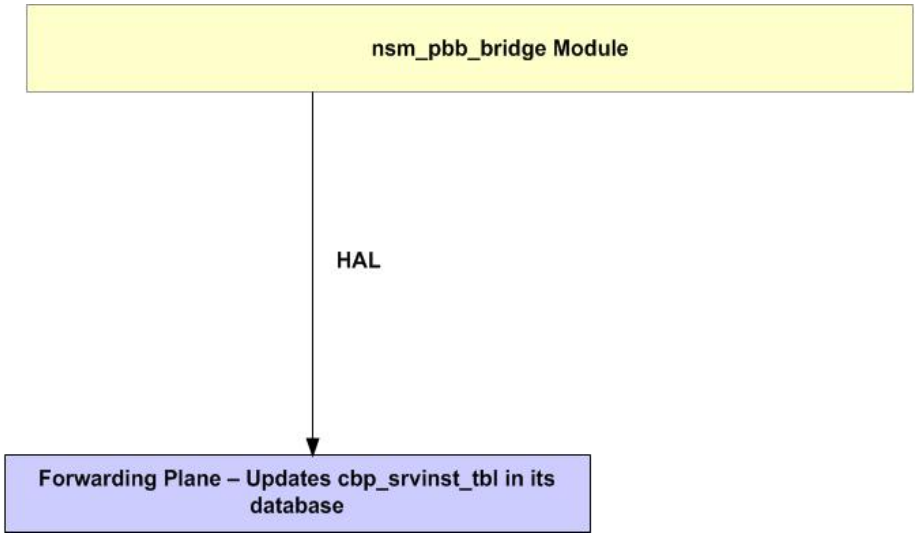


Figure 4-11: Dispatch Service Instance at Customer Backbone Port

Functions

The functions in this section are used for Provider Backbone Bridging.

Include File

To use the functions in this chapter, you must include nsm/L2/nsm_bridge_.h.

nsm_pbb_create_isid

This function is called to create an VIP node and in case of IB_BEB it is mapped to a PIP.

Syntax

```
int
nsm_pbb_create_isid (struct nsm_bridge_master *master, char *bridge_name,  u_int32_t
                     first, u_int32_t last, char *instance_name)
```

Input Parameters

`master`	Pointer to the bridge master structure
`bridge_name`	Pointer to the bridge instance associated
`instance_name`	Pointer to the service instance

Output Parameters

None

Return Value

RESULT_SUCCESS

nsm_pbb_delete_isid

This function is called delete ISID and all its associations.

Syntax

```
int
nsm_pbb_delete_isid (struct nsm_bridge_master *master, char *bridge_name,
                     u_int32_t first, u_int32_t last)
```

Input Parameters

`master`	Pointer to the bridge master structure
`bridge_name`	Pointer to the bridge instance associated
`instance_name`	Pointer to the service instance

Output Parameters

None

Return Value

RESULT_SUCCESS

nsm_vlan_port_beb_add_cnp_portbased

This function is called to add a portbased CNP instance.

Syntax

```
nsm_vlan_port_beb_add_cnp_portbased (struct interface *ifp,
                                     struct interface * ifp1,
                                     u_int32_t isid,
                                     u_char *instance_name)
```

Input Parameters

`ifp`	Pointer to the interface
`isid`	ISIS value
`instance_name`	Pointer to the service instance

Output Parameters

None

Return Value

RESULT_SUCCESS

nsm_vlan_port_beb_add_cnp_svlan_based

This function is called to add a VLAN based CNP instance.

Syntax

```
nsm_vlan_port_beb_add_cnp_svlan_based(struct interface *ifp,
                                      struct interface *ifp1,
                                      u_int32_t isid,u_char * instance_name,
                                      nsm_vid_t start_vid, nsm_vid_t end_vid)
```

Input Parameters

`ifp`	Pointer to the interface
`isid`	ISIS value
`instance_name`	Pointer to the service instance
`start_vid`	
`end_vid`	

Output Parameters

None

Return Value

RESULT_SUCCESS

nsm_vlan_port_beb_del_cnp_portbased

This function is called to delete a portbased CNP instance.

Syntax

```
nsm_vlan_port_beb_del_cnp_portbased(struct interface *ifp, u_int32_t isid)
```

Input Parameters

`ifp`	Pointer to the interface
`isid`	ISIS value

Output Parameters

None

Return Value

RESULT_SUCCESS

nsm_vlan_port_beb_del_cnp_svlan

This function is called to delete range of svlan mapping to ISID.

Syntax

```
nsm_vlan_port_beb_del_cnp_svlan(struct interface *ifp, u_int32_t isid,
                                nsm_vid_t start_vid, nsm_vid_t end_vid)
```

Input Parameters

`ifp`	Pointer to the interface
`isid`	ISIS value
`start_vid`	
`end_vid`	

Output Parameters

None

Return Value

RESULT_SUCCESS

nsm_vlan_add_beb_port

This function is called to set beb port with default VID.

Syntax

```
nsm_vlan_add_beb_port (struct interface *ifp, nsm_vid_t vid,
                       enum nsm_vlan_port_mode mode,
                       enum nsm_vlan_port_mode sub_mode,
                       bool_t iterate_members, bool_t notify_kernel)
```

Input Parameters

`ifp`	Pointer to the interface
`vid`	
`mode`	
`sub_mode`	

Output Parameters

None

Return Value

RESULT_SUCCESS

nsm_vlan_del_beb_port

This function is called to delete a VLAN from a beb port.

Syntax

```
nsm_vlan_del_beb_port (struct interface *ifp, nsm_vid_t vid,
                       enum nsm_vlan_port_mode mode,
                       enum nsm_vlan_port_mode sub_mode,
                       bool_t iterate_members, bool_t notify_kernel)
```

Input Parameters

`ifp`	Pointer to the interface

`vid`

`mode`

`sub_mode`

Output Parameters

None

Return Value

RESULT_SUCCESS

nsm_vlan_port_beb_add_cbp_srv_inst

This function is called to add map ISID to bvid on CBP and add default -da and ISID-mapping.

Syntax

```
nsm_vlan_port_beb_add_cbp_srv_inst (struct interface *ifp,
                                    u_int32_t first, u_int32_t last,
                                    u_int32_t isid_map, nsm_vid_t bvid,
                                    enum nsm_vlan_port_mode type,
                                    u_int8_t default_mac[], bool_t edge,
                                    enum isid_mode mode)
```

Input Parameters

`ifp`	Pointer to the interface
`isid_map`	
`bvid`	The VID of the B-VLAN

Output Parameters

None

Return Value

RESULT_SUCCESS

nsm_vlan_port_beb_delete_cbp_srv_inst

This function is called to delete ISID mapping on CBP

Syntax

```
nsm_vlan_port_beb_delete_cbp_srv_inst (struct interface *ifp,
                                       u_int32_t first, u_int32_t last)
```

Input Parameters

`ifp`	Pointer to the interface
`first`	
`last`	

Output Parameters

None

Return Value

RESULT_SUCCESS

nsm_vlan_port_add_pnp

This function is called to add BVID to PNP port.

Syntax

```
nsm_vlan_port_add_pnp(struct interface *ifp, nsm_vid_t bvid)
```

Input Parameters

`ifp`	Pointer to the interface
`bvid`	The VID of the B-VLAN

Output Parameters

None

Return Value

RESULT_SUCCESS

nsm_vlan_port_del_pnp

This function is called to delete BVID to PNP port.

Syntax

```
nsm_vlan_port_del_pnp (struct interface *ifp, nsm_vid_t bvid)
```

Input Parameters

`ifp`	Pointer to the interface
`bvid`	The VID of the B-VLAN

Output Parameters

None

Return Value

RESULT_SUCCESS

nsm_vlan_port_beb_pip_mac_update

This function is called to update the MAC address for the corresponding CBP.

Syntax

```
nsm_vlan_port_beb_pip_mac_update (struct interface *ifp, u_int8_t src_bmac[])
```

Input Parameters

`ifp`	Pointer to the interface
`bmac`	

Output Parameters

None

Return Value

RESULT_SUCCESS

nsm_vlan_port_config_vip

This function is called to configure a VIP node.

Syntax

```
nsm_vlan_port_config_vip(struct interface *ifp, u_int32_t isid,
                         u_char *instance_name, u_int8_t default_mac[],
                         enum nsm_port_service_type egress_type);
```

Input Parameters

`ifp`	Pointer to the interface
`isid`	
`instance name`	
`mac`	

Output Parameters

None

Return Value

RESULT_SUCCESS

nsm_pbb_check_service

This function is called to check the service instance status.

Syntax

```
nsm_pbb_check_service (struct interface *ifp,u_int32_t instance_id,
                       u_char *instance_name)
```

Input Parameters

`ifp`	Pointer to the interface
`instance_id`	
`instance name`	

Output Parameters

None

Return Value

RESULT_SUCCESS

nsm_pbb_dispatch_service

This function is called to dispatch the service instance to HAL.

Syntax

```
nsm_pbb_dispatch_service (struct interface *ifp, u_int32_t isid, u_char *name)
```

Input Parameters

`ifp`	Pointer to the interface
`isid`	
`name`	

Output Parameters

None

Return Value

RESULT_SUCCESS

nsm_pbb_remove_service

This function is called to remove the service instance from HA

Syntax

```
nsm_pbb_remove_service(struct interface *ifp, u_int32_t isid, u_char *name)
```

Input Parameters

`ifp`	Pointer to the interface
`isid`	
`name`	

Output Parameters

None

Return Value

RESULT_SUCCESS

Data Structure

The `nsm_beb_bridge {}` data structure supports PBB functionality. This structure contains Backbone Edge Bridge (BEB) bridge-level objects and tables. It also contains the I-component and B-component containers where service instance-related tables for every I-component bridge and a single B-component bridge are kept. Per 802.1ah, a BEB

bridge is considered as one bridge, which may contain multiple I-components and a single B-component. Each I- or B-component is in turn a bridge (S-VLAN component). For an I-BEB, the `B-Comp` container is NULL. For a B-BEB, the `I-Comp` container is `NULL`.

The `nsm_bridge_master` structure keeps a pointer to the same (the only) `nsm_beb_bridge` structure so that when an `nsm_bridge` is created, that pointer is used to reference the common `nsm_beb_bridge` tables. A maximum of 32 nsm_bridges, each representing an I-component is allowed. A single and independent `nsm_bridge` structure is maintained for a B-comp. Again, this `B-Comp` `nsm_bridge` also references the common `nsm_beb_bridge{}` structure via the `nsm_bridge_master` as shown in the figure

CFM in Provider Backbone Bridges

Connectivity Fault Management (CFM) in Provider Backbone Bridges (PBB) supports operation over LAN segments, Customer VLANs (C-VLAN), Service VLANs (S-VLAN), Backbone VLANs (B-VLAN), and backbone service instances identified by a Service Instance ID (I-SID). The CFM Maintenance End Points (MEP) and Maintenance Intermediate Points (MIP) may be inserted at any Enhanced Internal Services Sublayer (E-ISS) service interface. To comply with the IEEE 802.1ag 2007 standard, Multiple BVIDs per Maintenance Association (MA) and Link Level Maintenance Domains (MD) are supported for PBB. In addition, Default MD Level, MD and MA Name Format, MIP Creation permissions, and MEP Attributes are supported for Provider Backbone Bridges in ZebOS-XP.

Maintenance Domains

There are five distinct categories of maintenance domains (MD) used for CFM functions in PBB:

- Service VLAN ID domain
- Service Instance domain
- B-VLAN domain, LAN Links domain
- Link Level domain

S-VLAN Maintenance Domain

An S-VLAN MD is only visible within an I-component where customer frames are not encapsulated. Customer CFM frames are encapsulated along with other customer frames at Virtual Instance Ports (VIP) within the I-component. When S-VLAN CFM frames are encapsulated, they appear just like any data frame within a PBB network (PBBN). This means that they do not activate any CFM functions within the PBBN past the VIP.

Service Instance Domain

In a backbone Service Instance Domain, CFM frames are only visible within the PBBN where an I-TAG is processed. I-tags are processed at a Customer Backbone Port (CBP) and at a Provider Instance Port (PIP). The backbone service instance CFM maintenance domain may extend over E-NNI boundaries. There are eight levels that extend over all interconnected PBBNs until they terminate at a VIP.

B-VLAN Maintenance Domain

The B-VLAN maintenance domain manages B-VLANs within a single PBBN. A B-VLAN MD occurs once in each PBBN, and does not extend outside the boundaries of the PBBN.

LAN Links Domain

The LAN links within a PBBN, for example, a Provider Network Port (PNP) to a CBP, or a PIP to a CBP between two PBBNs (CBP-to-CBP or PIP-to-CBP), that enter or exit the PBBN (forming a CNP), may optionally be monitored by the LAN link MD. They can be generated or terminated within a PIP, a CBP, a PNP, or a Customer Network Port (CNP).

Link Level Maintenance Domain

Support for a link-level maintenance domain is provided as an extension to the link-level MD as defined in Chapter 2, *Connectivity Fault Management*. PBB commands and functions accept either, or both the ISID and VID passed by the operator. This means that the command can pass the VID as equal to 0 and the ISID as equal to 0 (VID=0 and ISID=0). When the MD is subsequently identified, based on the MD-type, it is determined whether the domain-type is link-level or not. Frame flow allowances for sending and receiving are the same as implemented for Link-Level MEP, also described in Chapter 2, *Connectivity Fault Management*. Untagged frames are sent and received irrespective of whether the port is an access port or a trunk port.

Multiple BVIDs per Maintenance Association

Connectivity Fault Management in Provider Backbone Bridges supports multiple BVID per BVLa MEP. This means that when a BVLAN domain is configured, multiple BVIDs can be monitored using a single MEP. However, on a CBP where Up MEP stacking takes place, the concept of a Primary ISID and a Secondary ISID is introduced.

Concepts of Primary and Secondary VIDs are introduced in Chapter 5. Each Service Domain MEP — either an Up MEP at a Customer Backbone Port or a Down MEP at a Provider Instance Port — is able to monitor multiple ISIDs. It can send frames only via the Primary ISID, but can receive frames from any Secondary ISID. At a CBP, it must be ensured that all secondary ISIDs, identified by a primary ISID, are mapped to the same BVID. Since ISID-to-BVID mapping information is managed in the NSM, the check for mapping Primary to Secondary ISIDs is located in the NSM module, and no error message is sent to the operator from the OAM Network Management Daemon (ONMD).

Connectivity Fault Management Operation

When CFM data originates from customer equipment and is received by a provider bridge, the C-VLAN aware bridge tags the CFM data from the customer with its VID (VLAN ID). When CFM data is received by an S-VLAN aware bridge, CFM encapsulation is not visible to the S-VAN aware bridge. Though the encapsulation is invisible, the MAC address are not, because they are determined by the MD level, which means that all addresses are in the same MD level.

CFM Operations in Provider Backbone Bridges

Four distinct categories of maintenance domains (MD) are used for CFM in PBB. These are the Service VLAN ID (S-VID) domain, Service Instance (I-SID) domain, B-VLAN domain and Link-Level domain.

1. An S-VLAN MD is only visible within an I-component where customer frames are not encapsulated. Customer CFM frames are encapsulated along with the other customer frames at the VIPs (virtual instance ports) within the I-component. Once S-VLAN CFM frames are encapsulated, they appear just like any data frame within the PBBN, therefore they do not activate any CFM functions within the PBBN past the VIP.
2. The backbone service instance CFM maintenance domain may extend over E-NNI boundaries. There are eight levels that extend over all interconnected PBBN until they are terminated at a VIP.
3. A B-VLAN MD manages the B-VLANs within a single PBBN. The B-VLAN MD is for each PBBN, and never extends past the PBBN boundaries.
4. A LAN link MD within a PBBN (PIP-to-CBP, PNP-to-PNP), between two PBBNs (CBP-to-CBP, PIP-to-CBP), or entering or egressing the PBBN (to or from a CNP) may optionally be monitored by a LAN link MD, and generated or terminated within a PIP, CBP, PNP or CNP. The LAN link MD is typically confined to the LAN link.

Maintenance Domains in PBB

There may be four maintenance domains per port, and the same port can be configured as a different CFM flow point in different MDs. A command exists for configuring a domain with an MD type. Based on the type of the domain configured, a MEP or MIP configured on the port is validated.

I-Component Maintenance Points

Connectivity fault management flow points can be configured for an I-component. Port-based and S-tagged service interfaces in the CNP and PIP need to support all the flow points.

Customer Network Port

On an I-component, Customer Network Ports (CNP) need to support VLAN-delimited DOWN MEP and MIP. A CNP port can be configured as a MIP per-SVID as shown in flow (`i`) above. A CNP can be configured as a DOWN MEP per-link domain as shown in (`j`) in the figure above.

Provider Instance Port

An I-component bridge PIP needs to support I-tag delimited DOWN MEP in backbone Service Instance (I-SID) domains. A PIP port can be configured as a DOWN MEP in both I-SID and Link-Level MDs as shown in flows (`f`) and (`c`) above. When using a Hierarchical Network-to-Network Interface (H-NNI), a PIP can be configured as an UP MEP in a B-VID domain. In the case of an I-tagged interface, an I-component PIP must support a link-level DOWN MEP without I-tags

B-Component Maintenance Points

The design for supporting CFM flow points for B-components is covered in this section.

Customer Backbone Port for I-Tagged Interface

In a B-component (backbone bridge), customer backbone ports must support I-tag delimited MIPs as shown in flow (`f`) above. The port type should be CFM_PBB_VLAN_PORT_MODE_CBP. A check is provided to determine whether the port can be configured as a MIP in this domain. The CLI command calls the function cfm_pbb_add_mip().

CBP for Interface Without I-Tag

A B-component CBP port must support a link-level Down MEP without an I-tag.

CBP in Peer E-NNI Service Interface

A B-component (backbone bridge) CBP port must support I-tag delimited DOWN MEP in the peer E-NNI service interface. The port type must be CFM_PBB_VLAN_PORT_MODE_CBP. Setting the optional keyword `enni` when configuring the CBP in the MD domain causes a check to validate whether the port may be configured as a MEP in this domain. The I-SID is validated for the MEP configured on this port.

CPB for B-Component

A B-component must support B-VLAN UP MEP (bridge level) at the CBP as shown in flow (`b`) above.Validation is performed to determine whether the B-VID has been configured on this CBP. A check is provided to verify that the bridge type is set to backbone.

CBP Per B-VLAN, Per I-SID

The B-component (backbone bridge) must support per B-VLAN, per I-SID UP MEP (bridge level) as shown in flow (`e`) above. This implies that the I-SID UP MEP is B-VLAN, or PNP-facing only. The MEP needs to be configured on a CBP. The uniqueness of this MEP is that it is per-instance, per-B-VLAN. It is set up to monitor a particular service instance carried within a particular B-VLAN. The identifiers for this class of MEP are B-VID, I-SID, and level. The commands used to make this configuration include the following:

```
ethernet cfm pbb mep up mpid MEPID domain-name NAME vid <1-4095> cbp-service-domain
  isid <1-16777214> backbone
```

```
ethernet cfm pbb mep up mpid MEPID domain-name NAME isid <0-16777214> backbone
```

Note that:

- The port type must be CFM_PBB_VLAN_PORT_MODE_CBP.
- A check is provided to validate whether the B-VID is configured on the CBP.
- A check is provided to validate whether the I-SID is mapped to this B-VID on the backbone bridge.
- A check is provided to verify that the bridge type is set to backbone.

Upon executing the first command, a MEP struct with type UP for B-VLAN MD on CBP is created and added to the MEP's list of cfm_port. Upon executing the second command, a MEP struct with type UP for SERVICE MD on CBP is created and added to MEP list of cfm_port. The MEP struct is added to the isid_up_mep_list of BVLAN UP MEP struct at the same level. An I-SID UP MEP is supported only when there is a B-VID UP MEP on that port in the same LEVEL, and the B-VID and the domain-type match. Otherwise, the API for the command returns an error saying that there is no B-VID UP MEP supported on that port.

B-Component Provider Network Port

B-component (backbone bridge) PNP ports must support MIP in a B-VID domain.

The command used is

```
ethernet cfm mip domain-name NAME backbone
```

- The port type needs to be CFM_PBB_VLAN_PORT_MODE_PNP.
- A check is provided to verify whether the domain type is BVLAN.
- A check is provided to verify whether the "backbone" bridge is selected.

Multiple BVID per MA

To provide the foundation for implementing the IEEE 802.1ag 2007 standard, sMultiple BVLAN IDs per MA are supported for PBB.

CFM-in-PBB Frame Handling

1. In a Service Domain, the functions sending out CFM frames, for example, `cfm_tx_lmm_frame()`, the first two fields of the PDU are Ethernet Type (two bytes - 0x88e7) and I-SID (four bytes), which identifies the frame as an I-tagged CFM frame.
2. Upon receiving a CFM frame, for the function `cfm_receive_pdu()`, if either the `HAVE_I_BEB or HAVE_B_BEB` flag is set, then the first two bytes from the payload are decoded and a check is provided for the Ethertype to identify whether it is an I-tagged CFM frame. If it is, then the port type must be PIP, CBP or CNP. Based port type, another function is called, `cfm_port_get_mep_by_isid()`.
3. The frame format for an I-tagged CFM frame that is transmitted or received is described below:

   B-VID — default VLAN configured on the PIP port

   B-DA — destination MAC address

   B-SA — source MAC address (of the source MEP)

   I-SID — present in the MEP structure

   Ethertype — the I-tag TPID. TPID is 0x88e7 from the first two-byte segment of any CFM PDU.

   ```
   | B-DA | B-SA | 88a8 | B-VID | payload |
   | Ethertype | I-SID | CFM PDU |
   ```

4. In a B-VID domain, the B-VID is similar to a VID.

   If a domain is E-NNI or H-NNI, and CBP is configured with UP B-VID MEP and UP I-SID MEP.

5. In an S-VID domain, the S-VID is similar to a VID. No ports except CNP participate in this domain, and CNP is a normal MIP in an S-VID domain.

SNMP API

The Provider Backbone Bridge SNMP support is implemented according to ieee8021PbbMib as described in the IEEE 802.1ap-2008 standard.

ieee8021PbbBackboneEdgeBridgeObjects

This group contains edge bridge configuration settings.

Object Type	Syntax	Access
ieee8021PbbBackboneEdgeBridgeAddress	MacAddress	read-only
ieee8021PbbBackboneEdgeBridgeName	SnmpAdminString	read-write
ieee8021PbbNumberOfIComponents	Unsigned32	read-only
ieee8021PbbNumberOfBComponents	Unsigned32	read-only
ieee8021PbbNumberOfBebPorts	Unsigned32	read-only

ieee8021IsidToVipTable

This group contains Service Instance IDs to Virtual Instance Ports (ISID-VIP) cross references for a bridge.

Object Type	Syntax	Access
`ieee8021PbbISidToVipComponentId`	IEEE8021PbbComponentIdentifier	read-only
`ieee8021PbbISidToVipPort`	IEEE8021BridgePortNumber	read-only

ieee8021PbbPipTable

These contains configuration settings for Provider Instance Ports (PIPs) on a bridge.

Object Type	Syntax	Access
ieee8021PbbPipBMACAddress	MacAddress	read-create
ieee8021PbbPipName	SnmpAdminString	read-create
ieee8021PbbPipIComponentId	IEEE8021PbbComponentIdentifier	read-only
ieee8021PbbPipVipMap	OCTET STRING	read-create

ieee8021PbbCBPServiceMappingTable

This group contains configuration settings for Customer Backbone Ports (CBPs) on a bridge.

Object Type	Syntax	Access
ieee8021PbbCBPServiceMappingBVid	VlanId	read-create
ieee8021PbbCBPServiceMappingDefaultBackboneDest	MacAddress	read-create
ieee8021PbbCBPServiceMappingType	IEEE8021PbbIngressEgress	read-create
ieee8021PbbCBPServiceMappingLocalSid	IEEE8021PbbServiceIdentifierOrUnassigned	read-create

ieee8021PbbVipToPipMappingGroup

This group contains VIP-PIP cross references for a bridge.

Object Type	Syntax	Access
ieee8021PbbVipToPipMappingPipIfIndex	InterfaceIndex	read-create
ieee8021PbbVipToPipMappingStorageType	StorageType	read-create

ieee8021PbbVipGroup

This group contains configuration settings for Virtual Instance Ports (VIPs) for a bridge.

Object Type	Syntax	Access
ieee8021PbbVipPipIfIndex	InterfaceIndexOrZero	read-only
ieee8021PbbVipISid	IEEE8021PbbServiceIdentifierOrUnassigned	read-create
ieee8021PbbVipDefaultDstBMAC	MacAddress	read-only
ieee8021PbbVipType	IEEE8021PbbIngressEgress	read-create
ieee8021PbbVipRowStatus	RowStatus	read-create

Customer Network Ports Group

This group contains configuration settings for Customer Network Ports (CNPs) for a bridge. This group is ZebOS-XP-specific and is not part of the standard.

Functions

The functions in this section are used for the SNMP support for Provider Backbone Bridging.

Function	Description
pbb_snmp_ahBridge_scalars	Reads a variable in ieee8021PbbBackboneEdgeBridgeObjects
pbb_snmp_ahCnp_table	Reads a variable in the Customer Network Port table
pbb_snmp_ahIsid_Vip	Reads a variable in ieee8021IsidToVipTable
pbb_snmp_ahPip_table	Reads a variable in ieee8021PbbPipTable
pbb_snmp_ahService_table	Reads a variable in ieee8021PbbCBPServiceMappingTable
pbb_snmp_ahVip_Pip_map	Reads a variable in ieee8021PbbVipToPipMappingGroup
pbb_snmp_ahVip_table	Reads a variable in ieee8021PbbVipGroup
pbb_snmp_cbp_lookup_by_isid	Finds an entry in the CBP table given an ISID
pbb_snmp_cnp_entry_add	Adds a entry to the Customer Network Port group
pbb_snmp_cnp_lookup_by_isid	Finds an entry in the Customer Network Port table given an ISID
pbb_snmp_pip_lookup_by_ifindex	Finds an entry in the PIP table given an interface index.
pbb_snmp_sid_vip_entry_add	Adds an entry to the ISID to VIP cross reference table
pbb_snmp_sid_vip_lookup_by_isid	Finds an entry in the ISID-VIP cross reference table given an ISID
pbb_snmp_vip_entry_add	Adds an entry to the VIP table
pbb_snmp_vip_lookup_by_portnum	Finds an entry in the VIP table given an port
pbb_snmp_vip_pip_lookup_by_portnum	Finds an entry in the VIP-PIP cross reference table given a port
pbb_snmp_write_bridge_name	Sets a variable in ieee8021PbbBackboneEdgeBridgeObjects
pbb_snmp_write_Cnp_rowstatus	Writes a row status entry in the Customer Network Port table
pbb_snmp_write_Cnp_table	Writes a entry in the Customer Network Port table.
pbb_snmp_write_pip_table	Sets a variable in ieee8021PbbPipTable
pbb_snmp_write_Service_table	Sets a variable in ieee8021PbbCBPServiceMappingTable
pbb_snmp_write_Vip_Pip_map	Sets a variable in ieee8021PbbVipToPipMappingGroup
pbb_snmp_write_Vip_Pip_Rowstatus	Sets a writes a row status for the VIP-PIP mapping table
pbb_snmp_write_vip_Rowstatus	Writes a row status in the VIP table
pbb_snmp_write_vip_table	Sets a variable in the ieee8021PbbVipGroup.

Include File

To use the functions in this chapter, you must include `nsm/L2/snmp/nsm_pbb_mib.h`.

pbb_snmp_ahBridge_scalars

This function is called when an external network management system queries an SNMP variable in the ieee8021PbbBackboneEdgeBridgeObjects table.

Syntax

```
u_char *
pbb_snmp_ahBridge_scalars (struct variable *vp,
                           oid * name,
                           size_t * length,
                           int exact, size_t * var_len,
                           WriteMethod ** write_method,
                           u_int32_t vr_id)
```

Input Parameters

`variable`	SNMP variable structure
`name`	OID name
`length`	OID length
`exact`	Exact
`write_method`	Write method pointer
`vr_id`	VR ID

Output Parameters

`var_len`	Length of the variable that was read

Return Value

A pointer to the value of the variable when the function succeeds

Zero (0) or NULL when the function fails

pbb_snmp_ahCnp_table

This function is called when an external network management system queries an variable in the Customer Network Port (CNP) table.

Syntax

```
unsigned char *
pbb_snmp_ahCnp_table (struct variable *vp, oid * name,
                      size_t * length, int exact, size_t * var_len,
                      WriteMethod ** write_method,
                      u_int32_t vr_id)
```

Input Parameters

`variable`	SNMP variable structure
`name`	OID name
`length`	OID length
`exact`	Exact match

size_t	Actual size
write_method	Write method pointer
vr_id	VR ID

Output Parameters

var_len	Length of the variable that was read

Return Value

A pointer to the value of the variable when the function succeeds

Zero (0) or NULL when the function fails

pbb_snmp_ahIsid_Vip

This function is called when an external network management system queries an SNMP variable in the ieee8021IsidToVipTable table.

Syntax

```
unsigned char *
pbb_snmp_ahIsid_Vip (struct variable *vp, oid * name,
                     size_t * length, int exact, size_t * var_len,
                     WriteMethod ** write_method,
                     u_int32_t vr_id)
```

Input Parameters

variable	SNMP variable structure
name	OID name
length	OID length
exact	Exact match
size_t	Actual size
write_method	Write method pointer
vr_id	VR ID

Output Parameters

var_len	Length of the variable that was read

Return Value

A pointer to the value of the variable when the function succeeds

Zero (0) or NULL when the function fails

pbb_snmp_ahPip_table

This function is called when an external network management system queries an SNMP variable in ieee8021PbbPipTable.

Syntax

```
unsigned char *
pbb_snmp_ahPip_table (struct variable *vp, oid * name,
                      size_t * length, int exact, size_t *var_len,
                      WriteMethod ** write_method, u_int32_t vr_id)
```

Input Parameters

`variable`	SNMP variable structure
`name`	OID name
`length`	OID length
`exact`	Exact match
`size_t`	Actual size
`var_len`	Length of OID
`write_method`	Write method pointer
`vr_id`	VR ID

Output Parameters

`var_len`	Length of the variable that was read

Return Value

A pointer to the value of the variable when the function succeeds

Zero (0) or NULL when the function fails

pbb_snmp_ahService_table

This function is called when an external network management system queries an SNMP variable in ieee8021PbbCBPServiceMappingTable.

Syntax

```
unsigned char *
pbb_snmp_ahService_table (struct variable *vp, oid * name,
                          size_t * length, int exact,
                          size_t * var_len,
                          WriteMethod ** write_method,
                         u_int32_t vr_id)
```

Input Parameters

`variable`	SNMP variable structure
`name`	OID name
`length`	OID length
`exact`	Exact match
`size_t`	Actual size
`var_len`	Length of OID
`write_method`	Write method pointer
`vr_id`	VR ID

Output Parameters

`var_len`	Length of the variable that was read

Return Value

A pointer to the value of the variable when the function succeeds

Zero (0) or NULL when the function fails

pbb_snmp_ahVip_Pip_map

This function is called when an external network management system queries an SNMP variable in ieee8021PbbVipToPipMappingGroup.

Syntax

```
unsigned char *
pbb_snmp_ahVip_Pip_map (struct variable *vp, oid * name,
                        size_t * length, int exact,
                        size_t * var_len,
                        WriteMethod ** write_method,
                        u_int32_t vr_id)
```

Input Parameters

`variable`	SNMP variable structure
`name`	OID name
`length`	OID length
`exact`	Exact match
`size_t`	Actual size
`write_method`	Write method pointer
`vr_id`	VR ID

Output Parameters

`var_len`	Length of the variable that was read

Return Value

A pointer to the value of the variable when the function succeeds

Zero (0) or NULL when the function fails

pbb_snmp_ahVip_table

This function is called when an external network management system queries an SNMP variable in ieee8021PbbVipGroup.

Syntax

```
unsigned char *
pbb_snmp_ahVip_table (struct variable *vp, oid * name,
                      size_t * length, int exact,
                      size_t * var_len,
```

```
                    WriteMethod ** write_method,
                    u_int32_t vr_id)
```

Input Parameters

`variable`	SNMP variable structure
`name`	OID name
`length`	OID length
`exact`	Exact match
`size_t`	Actual size
`write_method`	Write method pointer
`vr_id`	VR ID

Output Parameters

`var_len`	Length of the variable that was read

Return Value

A pointer to the value of the variable when the function succeeds

Zero (0) or NULL when the function fails

pbb_snmp_cbp_lookup_by_isid

This function finds an entry in the CBP table given an ISID.

Syntax

```
struct cbp_srvinst_tbl*
pbb_snmp_cbp_lookup_by_isid (struct avl_tree *tree, u_int32_t b_isid,
                             int exact)
```

Input Parameters

`svl_tree`	AVL tree structure
`b_isid`	Service instance ID value
`exact`	Value true or false

Output Parameters

None

Return Value

Pointer to `cbp_srvinst_tbl` structure when the function succeeds

pbb_snmp_cnp_entry_add

This function adds a entry to the Customer Network Port (CNP) group.

Syntax

```
int
```

```
pbb_snmp_cnp_entry_add(u_int32_t icomp_id, u_int32_t isid,
                            u_int32_t base_port)
```

Input Parameters

`icomp_id`	I-component ID
`isid`	Service instance ID (ISID)
`base_port`	Base port number

Output Parameters

None

Return Value

Integer value that denotes whether the function succeeded or failed

pbb_snmp_cnp_lookup_by_isid

This function finds an entry in the Customer Network Port (CNP) table given an ISID.

Syntax

```
struct cnp_srv_tbl *
pbb_snmp_cnp_lookup_by_isid (struct avl_tree *tree,
                             u_int32_t isid, int exact)
```

Input Parameters

`svl_tree`	AVL tree structure
`isid`	Service instance ID value
`exact`	Exact value; true or false

Output Parameters

None

Return Value

Pointer to `pip_tbl` structure when the function succeeds

pbb_snmp_pip_lookup_by_ifindex

This function finds an entry in the PIP table given an interface index.

Syntax

```
struct pip_tbl *
pbb_snmp_pip_lookup_by_ifindex (struct list *list, u_int32_t ifindex, int exact)
```

Input Parameters

`list`	List structure
`ifindex`	Interface index
`exact`	Value true or false

Output Parameters

None

Return Value

Pointer to `pip_tbl` structure when the function succeeds

pbb_snmp_sid_vip_entry_add

This function adds an entry to the ISID to VIP cross reference table.

Syntax

```
int
pbb_snmp_sid_vip_entry_add (u_int32_t icomp_id, u_int32_t vip_isid)
```

Input Parameters

`icomp_id`	I-component ID
`vip_isid`	VIP-to-ISID value.

Output Parameters

None

Return Value

Integer value that informs whether the ISID_VIP cross reference entry creation process succeeded or failed

pbb_snmp_sid_vip_lookup_by_isid

This function finds an entry in the ISID-VIP cross reference table given an ISID.

Syntax

```
struct sid2vip_xref *
pbb_snmp_sid_vip_lookup_by_isid (struct avl_tree *tree, u_int32_t isid, int exact)
```

Input Parameters

`avl_tree`	AVL tree structure
`isid`	ISID value
`exact`	Wether the value is an exact match; true or false

Output Parameters

None

Return Value

Pointer to `sid2vip_xref` structure when the function succeeds

pbb_snmp_vip_entry_add

This function adds an entry to the VIP table.

Syntax

```
int
pbb_snmp_vip_entry_add(u_int32_t icomp_id, u_int32_t base_port)
```

Input Parameters

`icomp_id`	I-component ID
`base_port`	Base port number

Output Parameters

None

Return Value

Integer value that informs whether the VIP entry creation success or failed.

pbb_snmp_vip_lookup_by_portnum

This function finds an entry in the VIP table given an port.

Syntax

```
struct vip_tbl *
pbb_snmp_vip_lookup_by_portnum (struct avl_tree *tree, u_int32_t port_num,
                                int exact)
```

Input Parameters

`avl_tree`	AVL tree structure
`port_num`	Base port number
`exact`	Exact value; true or false

Output Parameters

None

Return Value

Pointer to `vip_tbl` structure when the function succeeds

pbb_snmp_vip_pip_lookup_by_portnum

This function finds an entry in the VIP-PIP cross reference table given a port.

Syntax

```
struct vip2pip_map *
pbb_snmp_vip_pip_lookup_by_portnum (struct avl_tree *tree, u_int32_t
                                    port_num, int exact)
```

Input Parameters

`tree`	AVL tree structure
`port_num`	Base port number

`exact`	Value true or false

Output Parameters

None

Return Value

Pointer to `vip2pip_map` structure when the function succeeds

pbb_snmp_write_bridge_name

This function is called when an external network management system sets an SNMP variable in ieee8021PbbBackboneEdgeBridgeObjects.

Syntax

```
int
pbb_snmp_write_bridge_name (int action, u_char * var_val,
                                        u_char var_val_type, size_t var_val_len,
                                        u_char * statP, oid *name, size_t name_len,
                                        struct variable *v, u_int32_t vr_id)
```

Input Parameters

`var_val_type`	Variable value type
`var_val_len`	Variable value length
`size_t`	Size of the value to be set
`name`	OID name
`name_len`	OID length
`variable`	SNMP variable structure.

Output Parameters

None

Return Value

Integer value that denotes whether the function succeeded or failed

pbb_snmp_write_Cnp_rowstatus

This function writes a row status entry in the Customer Network Port (CNP) table.

Syntax

```
int
pbb_snmp_write_Cnp_Rowstatus (int action, u_char * var_val,
                              u_char var_val_type, size_t var_val_length,
                              u_char * statP, oid * name, size_t name_len,
                               struct variable *v, u_int32_t vr_id)
```

Input Parameters

`var_val`	Value to be set in the `snmpset` request

var_val_type	Type of the value to be set
var_val_length	Size of the value to be set
name	OID name
size	OID length
variable	SNMP variable structure

Output Parameters

None

Return Value

Integer value which tells if the function succeeded or failed

pbb_snmp_write_Cnp_table

This function writes a entry in the Customer Network Port (CNP) table.

Syntax

```
int
pbb_snmp_write_Cnp_table (int action,
                          u_char * var_val,
                          u_char var_val_type,
                          size_t var_val_len,
                          u_char * statP, oid * name,
                          size_t name_len,
                          struct variable *v, u_int32_t vr_id)
```

Input Parameters

var_val	Value to be set in the snmpset request
var_val_type	Type of the value to be set
var_val_length	Size of the value to be set
name	OID name
size	OID length
variable	SNMP variable structure

Output Parameters

None

Return Value

Integer value that denotes whether the function succeeded or failed

pbb_snmp_write_pip_table

This function is called when an external network management system sets an SNMP variable in ieee8021PbbPipTable.

Syntax

```
int
pbb_snmp_write_pip_table (int action, u_char * var_val,
                          u_char var_val_type,
                          size_t var_val_len, u_char * statP, oid * name,
                          size_t name_len, struct variable *v, u_int32_t vr_id)
```

Input Parameters

`var_val`	Value to be set in the `snmpset` request
`var_val_type`	Type of the value to be set
`var_val_len`	Size of the value to be set
`name`	OID name
`name_len`	OID length
`variable`	SNMP variable structure.

Output Parameters

None

Return Value

Integer value that denotes whether the function succeeded or failed

pbb_snmp_write_Service_table

This function is called when an external network management system sets an SNMP variable in ieee8021PbbCBPServiceMappingTable.

Syntax

```
int
pbb_snmp_write_Service_table (int action, u_char * var_val,
                              u_char var_val_type, size_t var_val_len,
                              u_char * statP, oid * name, size_t name_len,
                              struct variable *v, u_int32_t vr_id)
```

Input Parameters

`var_val`	Value to be set in the `snmpset` request
`var_val_type`	Type of the value to be set
`var_val_length`	Size of the value to be set
`name`	OID name
`size`	OID length
`variable`	SNMP variable structure

Output Parameters

None

Return Value

Integer value that denotes whether the function succeeded or failed

pbb_snmp_write_Vip_Pip_map

This function is called when an external network management system sets an SNMP variable in the ieee8021PbbVipToPipMappingGroup.

Syntax

```
int
pbb_snmp_write_Vip_Pip_map (int action, u_char * var_val,
                            u_char var_val_type,
                            size_t var_val_len, u_char * statP,
                            oid * name, size_t name_len,
                            struct variable *v, u_int32_t vr_id)
```

Input Parameters

`var_val`	Value to be set in the `snmpset` request
`var_val_type`	Type of the value to be set
`var_val_length`	Size of the value to be set
`name`	OID name
`size`	OID length
`variable`	SNMP variable structure

Output Parameters

None

Return Value

Integer value that denotes whether the function succeeded or failed

pbb_snmp_write_Vip_Pip_Rowstatus

This function writes a row status for the VIP-PIP mapping table.

Syntax

```
int
pbb_snmp_write_Vip_Pip_Rowstatus (int action, u_char * var_val,
                                  u_char var_val_type, size_t var_val_len,
                                  u_char * statP, oid * name,
                                  size_t name_len, struct variable *v,
                                  u_int32_t vr_id)
```

Input Parameters

`var_val`	Value to be set for the `snmpset` request
`var_val_type`	Type of the value to be set
`var_val_len`	Size of the value to be set

`name`	OID name
`name_len`	OID length
`variable`	SNMP variable structure.

Output Parameters

None

Return Value

Integer value that denotes whether the function succeeded or failed

pbb_snmp_write_vip_Rowstatus

This function writes a row status in the VIP table.

Syntax

```
int
pbb_snmp_write_vip_Rowstatus (int action, u_char * var_val,
                              u_char var_val_type, size_t var_val_len,
                              u_char * statP, oid * name, size_t name_len,
                              struct variable *v, u_int32_t vr_id)
```

Input Parameters

`var_val`	Value to be set in the `snmpset` request
`var_val_type`	Type of the value to be set
`var_val_len`	Size of the value to be set
`name`	OID name
`name_len`	OID length
`variable`	SNMP variable structure

Output Parameters

None

Return Values

Integer value that denotes whether the function succeeded or failed

pbb_snmp_write_vip_table

This function is called when an external network management system sets an SNMP variable in the ieee8021PbbVipGroup.

Syntax

```
int
pbb_snmp_write_vip_table (int action, u_char * var_val,
                          u_char var_val_type,
                          size_t var_val_len, u_char * statP, oid * name,
                          size_t name_len, struct variable *v, u_int32_t vr_id)
```


Input Parameters

`var_val`	Value to be set in the `snmpset` request
`var_val_type`	Type the value to be set
`var_val_len`	Size of the value to be set
`name`	OID name
`name_len`	OID length
`variable`	SNMP variable structure

Output Parameters

None

Return Value

Integer value that denotes whether the function succeeded or failed

CHAPTER 5 Provider Backbone Bridge - Traffic Engineering

ZebOS-XP supports Provider Backbone Bridge - Traffic Engineering (PBB-TE) as specified in the IEEE 802.1Qay standard. The ZebOS-XP PBB-TE features for Connectivity Fault Management (CFM) protocols and addressing of PBB-TE Maintenance End Points (MEP), Maintenance Intermediate Points (MIP), and TE Service Instances (TESI) are included in this chapter.

Features

ZebOS-XP PBB-TE supports the following key features:

- Bridge management
- Basic filtering services
- Addressing services
- VLAN support
- Remote management protocol support
- Configuration by an external agent to provide TE Service Instances (TESI)
- Point-to-point TESI
- Point-to-multipoint TESI
- Configuration of a bridge to operate as a Backbone Core Bridge (BCB) that supports TESI
- Configuration of a bridge to operate as a Backbone Edge Bridge (BEB) that supports TESI
- Connectivity Fault Management (CFM)
- Protection switching
- Protection switching with load sharing

Architecture

The PBB-TE architecture supports IEEE 802.1ay PBB-TE, PBB-TE CFM, and PBB-TE Protection Switching (PS). Figure 5-1 shows the high-level ZebOS-XP Layer 2 switching architecture.

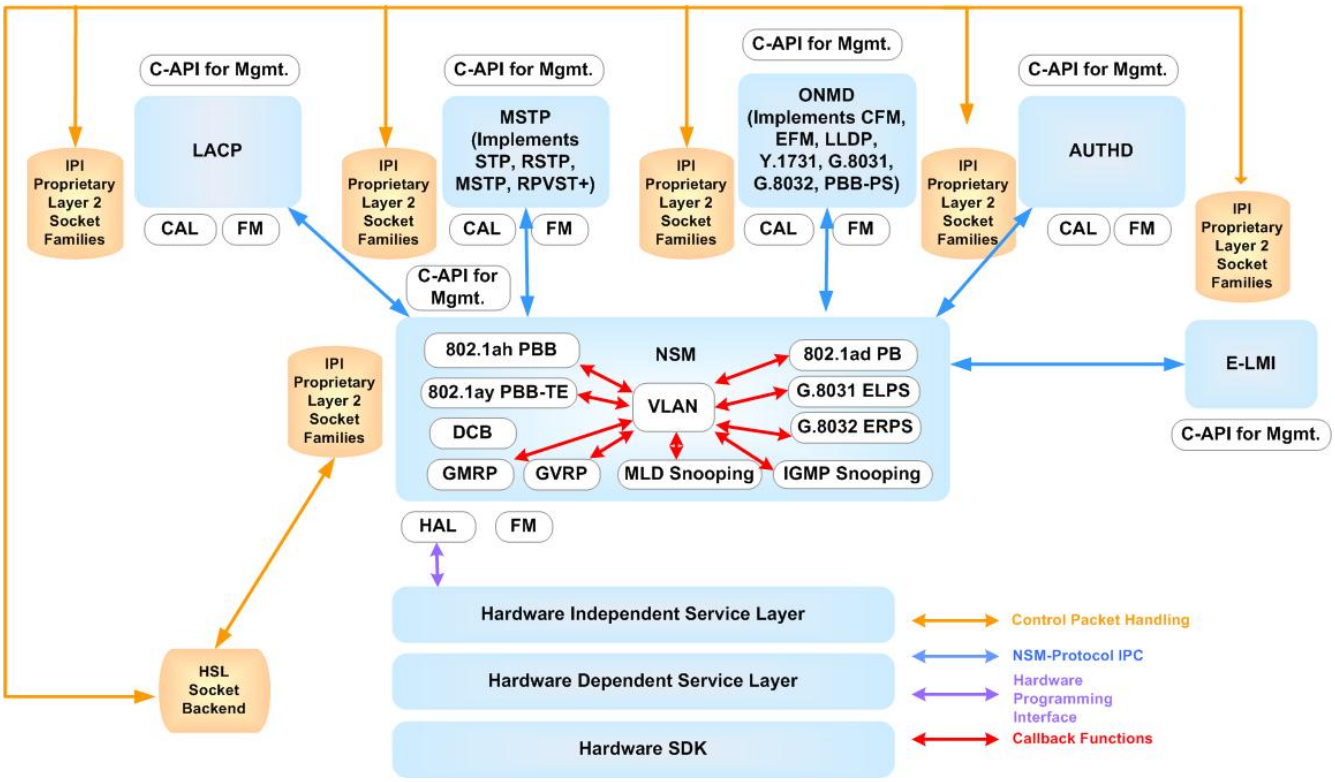


Figure 5-1: ZebOS-XP Layer 2 architecture

PBB-TE Region

A PBB-TE region is a set of IB-BEBs and BCBs capable of supporting TESI that are interconnected by a set of B-VLANs allocated to provisioned Ethernet Switched Paths.

Ethernet Switched Path

An Ethernet Switched Path (ESP) is a point-to-point or a point-to-multipoint switched Ethernet connection within a PBB-TE region. For a point-to-point connection, there is an explicit path; for a point-to-multipoint connection, there is a set of explicit paths. Each ESP can be identified by a 3-tuple, defined as follows:

```
<ESP-MAC DA, ESP-MAC SA, ESP-VID>
```

- ESP-MAC DA is the address of the Provider Instance Port (PIP) encapsulating the customer service instance that by configuration is the same as the address of the internally-connected source Customer Backbone Port (CBP).
- The ESP-MAC SA identifies CBP destination address(es).
- An ESP-VID is the VID value that distinguishes among ESPs having the same `<destination-source>` pair. It can only take values that are allocated to the PBB-TE Region identified by the TE-MSTID (`0xFFE`).

ESP-VID

An ESP-VID is the VLAN ID that is used as part of the ESP identifier.

TE-MSTID

A TE-MST is a special value (`0xFFE`) of the Multiple Spanning Tree (MST) ID in the MST Configuration Table. A TE-MSTID identifies an MST instance that is not controlled by the spanning tree protocol.

Point-to-Point ESP

A point-to-point ESP is a provisioned traffic-engineered, unidirectional connectivity path from one CBP to another that extends over a PBB Network (PBBN). The path is identified by the 3-tuple <ESP-MAC DA, ESP-MAC SA, ESP-VID> where:

- ESP-MAC DA is an individual MAC address
- ESP-MAC SA is an individual MAC address
- ESP-VID is allocated to the TE-MSTID

Point-to-Multipoint ESP

A point-to-multipoint ESP is a provisioned, traffic-engineered unidirectional connectivity path from one CBP to n CBPs that extends over a PBBN. The path is identified by the 3-tuple <ESP-MAC DA, ESP-MAC SA, ESP-VID> where:

- ESP-MAC DA is a group MAC address identifying *n* CBPs,
- ESP-MAC SA is an individual MAC address
- ESP-VID value is allocated to the TE-MSTID

Although an ESP is identified by the 3-tuple <ESP-MAC DA, ESP-MAC SA, ESP-VID>, it is only the {ESP-MAC DA-ESP-VID} pair that are used for forwarding decisions, and the ESP-VID determines the actual path taken through the network, while the ESP-MAC DA determines the destination CBP port. This means that within a PBB-TE region, there can be a total of 212 unique routing trees to any single destination CBP. These 212 unique routing trees must be shared by all source CBP that need to reach a specific destination CBP.

TE Service Instance

A TE service instance (TESI) is an instance of the MAC service provided by a set of ESPs, collectively identified by a single TE-SID that form a bidirectional service. Two types of TE service instances are defined - a point-to-point or a point-to-multipoint TE service instance.

Point-to-Point TE Service Instance

A point-to-point (PtP) TESI is an instance of the MAC service provided by two co-routed point-to-point ESPs and a pair of 3-tuples <DA1, SA1, VID1>, <SA1, DA1, VID2>, that form a bidirectional service where the endpoints of the ESPs have the same CBP MAC address.

Point-to-Multipoint TE Service Instance

A point-to-multipoint TESI is an instance of the MAC service provided by a set of ESPs that comprises one point-to-multipoint ESP from a root to each of *n* leaves, plus one unidirectional point-to-point ESP, routed from each of the *n* leaves to the root along the branches of the point-to-multipoint ESP. A Point-to-Multipoint (PtMP) TESI is provided by one multipoint ESP plus *n* unidirectional point-to-point ESPs that are routed along the leaves of the multicast ESPs and identified by *n*+1 3-tuples (<DA, SA, VID>, <SA, SA1, VID1>, <SA, SA*n*, VID*n*>). The Destination Address used by the

root CBP identifies the list of MAC addresses {SA1, SA2, through SA*n*}. A PtMP TESI is identified collectively by an implementation-dependent TES-ID (TE service identifier).

TE-SID

An implementation-dependent identifier of a PBB-TE service instance, which is formally identified by a series of 3-tuples <ESP-MAC DA, ESP-MAC SA, ESP-VID>. The TESI identifier should not be confused with an I-SID, and it is not used as tag information. There is a system-wide (per-BEB) one-to-one mapping between a unique 3-tuple and a unique TE-SID.

TE Protection Group

A TE protection group consists of a pair of point-to-point TESI terminating at the same pair of BEB CBPs, where only one TE service instance is used to carry traffic at a time.

Operation

A PBB-TE Domain is identified by a Maintenance Domain (MD) name and a level. A PBB-TE Maintenance Association (MA) is identified by a Traffic Engineering Service Instance ID and an MA ID.

A Maintenance End Point (MEP) is associated with exactly one MA. It is identified for management purposes by two parameters:

- A Maintenance Association Identifier (MAID)
- A Maintenance End Point Identifier (MEPID) assigned to the MEP

A MEP is identified for data forwarding purposes by a set of 3-tuples, <MAC-ESP DA, MAC-ESP SA, ESP-VID>, that is inherited from the MA. A PBB-TE MA requires a list of the component Ethernet Switched Paths (ESP), each identified by the 3-tuple <ESP-MAC DA, ESP-MAC SA, ESP-VID>, or collectively by the TE-SID.

Connectivity Fault Management in PBB-TE

Connectivity Fault Management may operate over LAN segments, Customer VLANs (C-VLAN), Service VLANs (S-VLAN), Backbone VLANs (B-VLAN), and backbone service instances identified by Service Instance IDs (I-SID). A PBB-TE Maintenance Association (MA) is operated over a TE Service instance ID (TE-SID), which is a 3-tuple (triplet) identified by < ESP-MAC DA, ESP-MAC SA, ESP-VID>.

The messages defined in the ITUT Y.1731 standard for Maintenance Communication Channel (MCC), Delay Measurement (1DM and DMM), Loss Measurement Messages (LMM), Alarm Indicator Signaling (AIS), Lock (LCK) and Test (TST) are also supported in ZebOS-XP PBB-TE.

Addressing PBB-TE MEPs

Configuration of a PBB-TE MA requires a parameter (or a list of parameters) that associates it with a monitored service. A TE service instance (TESI) is identified by a list of ESPs supporting it. The MEPs related to a TESI require the same set of parameters as VLAN-based MEPs require, with the following exceptions:

- A Primary VID is not writable, but is always set to the value of the ESP-VID parameter from the maintenance association’s component ESP that has the maintenance end point’s MAC address in its ESP-MAC SA field.
- MAC address of the MEP is the MAC address of the CBP port on which the MEP is operating.

PBB-TE MEPs are always UP MEPs and can only be placed on CBPs as they correspond to the demarcation points of the ESPs providing the PBB-TE service. PBB-TE MEPs have a Primary VID from which frames can egress, but a PBB-TE MEP can receive frames from any of the multiple VIDs in the TESI.

Point-to-Point PBB-TE MEP

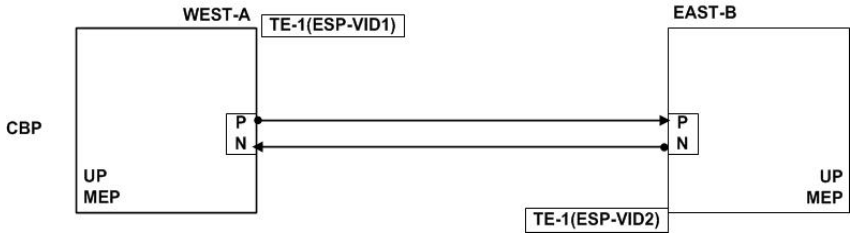


Figure 5-2: Point-to-Point PBB-TE MEP

In this diagram, TE-1 at WEST-A has ESPs as described below. The 3-tuple for CBP-A is:

- for ESP-1 <ESP-DA = CBP-B MAC, ESP-SA = CBP-A MAC, ESP-VID = 1>
- for ESP-2 <ESP-DA = CBP-A MAC, ESP-SA = CBP-B MAC, ESP-VID = 2>

For ESP-1, the ESP-SA is the same as the MAC for CBP-A MAC, so ESP-VID 1 is chosen as Primary VID for the UP MEP at WEST-A.

Note: As illustrated, the PBB-TE UP MEPs are located on the CBPs based on the TE-SID.

Since PBB-TE supports only IB-BEBs, a logical CBP port is created and added to the port list, thus permitting multiple TE-SIDs. In onmd, the logical CBP Port is added as an instance of the onmd_ifp, in which the list of TE-SIDs for that CBP is maintained. Each TE-SID has a list of ESPs identified by the 3-tuple < ESP-VID, ESP-DA, ESP-SA>.

A PBB-TE MEP may monitor multiple VLANs, or all of the ESP-VIDs in a TESI. Frames can egress only from the PBB-TE MEP via the Primary VID, but can ingress via any of the ESP-VIDs in the TESI.

The Primary VID is the ESP-VID for which the ESP-SA is the same as the CBP MAC. For all other ESPs, the ESP-DA is the CBP's MAC and ESP-SA is the remote MAC. The CBP at WEST-A has as its primary VID ESP-VID1, and the UP MEP at EAST-B has its primary VID as ESP-VID2.

Point-to-Multipoint PBB-TE MEP

For point-to-multipoint, or multicast, there is one ESP from the list of the ESPs in the TE-SID MA that is used to send a point-to-multipoint frame, and *n* number of paths from *n* leaves to the root. Point-to-multipoint ESPs keep a list of DAs (CBP-B, CBP-C, etc., through CBP-*n*).

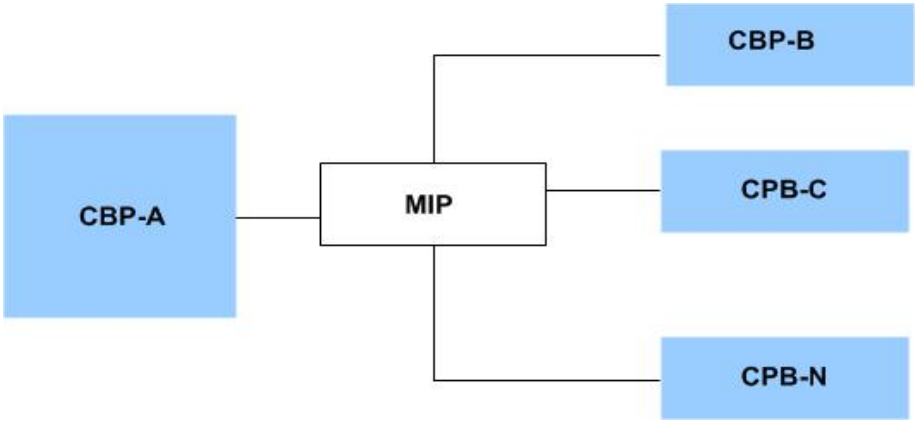


Figure 5-3: Point-to-Multipoint PBB-TE Instance

There is one ESP identified by the three tuple as:

- ESP-1 ESP-VID = 1
- ESP-SA = CBP-A MAC
- ESP-DA = Group MAC

The remaining ESPs for CBP-A include the following:

- ESP-2 <ESP-VID = 2, ESP-DA = CBP-A MAC, ESP-SA = CBP-B MAC>
- ESP-3 <ESP-VID = 3, ESP-DA = CBP-A MAC, ESP-SA = CBP-C MAC>
- ESP-*N* <ESP-VID = *N*, ESP-DA = CBP-A MAC, ESP-SA = CBP-*n* MAC>

In the case of a point-to-multipoint TESI, there is a point-to-multipoint ESP path identified by a 3-tuple <ESP-MAC DA, ESP-MAC SA, ESP-VID>, where the ESP-MAC DA is a group MAC address identifying *n* CBPs.

Continuity Check Messaging in a PBB-TE MA

In the case of a PBB-TE associated MEP, the destination_address parameter is set to the MAC address indicated by the ESP-MAC DA field of the ESP having in its ESP-MAC SA field the MAC address of the MEP.

Traffic field. In a PBB-TE MA, the second most significant bit of the Flags field is the Traffic bit. If supported, this bit is used to indicate the presence of backbone service instances in the monitored TE service instance. This bit is set to 1 if the transmitting MEP's presentTraffic variable is set, and 0 if it is not. This field is not examined when the receiving Maintenance Points are not PBB-TE MEPs supporting the traffic field.

Continuity Check Message

MEP ID. MEP ID is a 2-octet field where the 13 least-significant bits are used to identify the MEP transmitting the CCM frame. The MEP ID is unique within the Maintenance Entity Group (MEG).

Remote Defect Identification

RDI (Remote Defect Identification) is a 1-bit information element carried in the most significant bit of the Flags field. When the RDI bit is 1, detection of a defect is indicated by the transmitting MEP. When the RDI bit is 0, no defect indication is communicated by the transmitting MEP.

CCM Support for TE Traffic Field

trafficField Traffic field is a 1-bit information element carried in the second most significant bit of the Flags field. Traffic field is set to 1 when the presentTraffic variable is set. The presentTraffic variable is 1 for working-path MEPS and 0 when Protection Path is used.

presentTraffic This is a Boolean value indicating the state of the traffic bit in CCMs transmitted by a MEP. The presentTraffic value is TRUE if the BSI table of the CBP associated with the PBB-TE MEP contains an entry that has in its B-VID and Default Backbone Destination fields the values of ESP-VID and ESP-MAC DA corresponding to the 3-tuple identifier of the TESI's component ESP, which has in its ESP-MAC SA field the MAC address of the PBB-TE MEP.

This is set to 1 for all the PBB-TE MEPs for all active-path ESP entities, and it is set to 0 for all the PBB-TE MEPs for the inactive ESP entities.

rcvdTrafficBit This is a Boolean flag set by the MEP according to the trafficField of the last-received valid CCM.

Loopback Protocol in a PBB-TE MA

Transmitting Loopback Messages

1. The Loopback Messages (LBM) transmitted by a MEP associated with a PBB-TE MA use as the destination_address parameter the MAC address indicated by the value of the ESP-MAC DA field of the MA's component ESP that has the MEP's MAC address indicated in its ESP-MAC SA field.
2. To enable intermediate MIPs to selectively intercept LBMs that are targeting them, the PBB-TE MIP TLV is inserted as the first TLV in an LBM. The PBB-TE MIP TLV is not included if the LBM destination address is associated with any of the values in the ESP-MAC DA field of the monitored MA's component ESPs.

The format of PBB-TE MIP TLV is as follows

- **Type = 9**. Used only in MAs monitoring PBB-TE services. It enables intermediate MIPs to selectively intercept LBMs that are targeting them. The PBB-TE MIP TLV is not used if the target address of the LBM is associated with any of the values in the ESP-DA field of the monitored MA’s component ESPs, but otherwise it shall be present
- **MIP MAC address**. The MAC address of the MIP to which the LBM is targeted.
- **Reverse VID**. The parameter provided which is used as the vlan_identifier in the Loopback Response (LBR).
- **Reverse MAC**. This field is only included if the MEP is associated with a point-to-multipoint TE service instance. If the MEP is the root MEP, the Reverse MAC field contains the ESP-MAC SA value of any of the point-to-point ESPs in the TE service instance. If the MEP is a leaf MEP, the Reverse MAC field contains the ESP-MAC DA of the point-to-multipoint ESP in the TE service instance.

Validity Test

LBMs received by PBB-TE MEPs are not processed and are discarded if the received LBM carries a PBB-TE MIP TLV. A PBB-TE MIP forwards all received LBMs, except those carrying a PBB-TE MIP TLV containing in their MIP MAC address field the MAC address of the MIP associated with that PBB-TE MIP.

Loopback Response

1. The destination_address parameter used by Loopback Responses transmitted by PBB-TE MEPs is the value of the ESP-MAC DA of the component ESP that has the MEP's own address in its ESP-MAC SA field.
    - In the case of a PBB-TE MIP, the destination_address parameter for the transmitted LBR is the value carried in the Reverse MAC field contained in the PBB-TE MIP TLV of the received LBR, if the ESP-MAC DA in the LBM is a Group MAC address, otherwise the value of the source_address of the received LBM.
2. The source_address of LBRs transmitted by PBB-TE MIPs is set to:
    - the destination_address of the received LBM if this destination address is an individual MAC address, or
    - the value carried in the Reverse MAC field contained in the PBB-TE MIP TLV of the received LBM's destination address is a Group MAC address.

The vlan_identifier used by LBRs transmitted by PBB-TE MEPs is set to the value of its Primary VID. The vlan_identifier used by LBRs transmitted by PBB-TE MIPs is set to the value carried in the Reverse VID field contained in the PBB-TE MIP TLV of the received LBM.

LBM/LBR Frame Flow

In the figure below, consider a point-to-multipoint LBM.

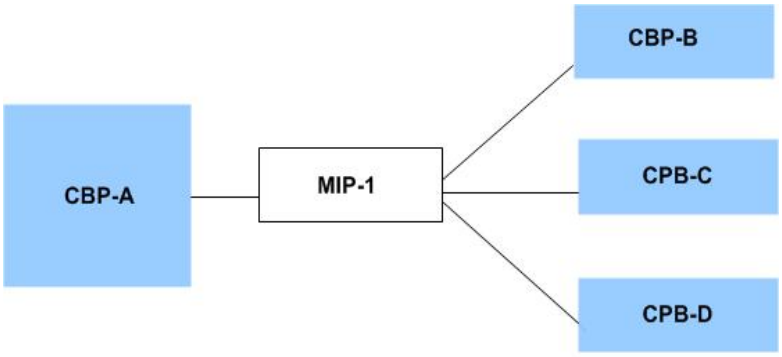


Figure 5-4: Point-to-Multipoint LBM

Note: When a reply is requested from a MIP, only then is the PBB-TE MIP TLV added to the frame.

Root Initiates Point-to-multipoint LBM

When CBP-A (Root MEP) sends an LBM with the group MAC, the PBB-TE MIP TLV contains

- Type = 9
- Length = 14
- Reverse VID = One from the ESP list of the TE-SID, used only by the MIP
- MIP MAC Address = MIP MAC
- Reverse MAC = ESP-SA of CBP-A

Because the MIP MAC field matches the MIP-1 MAC, the reply is sent from MIP-1, and the LBM is not forwarded. In this case, the PBB-TE MEPs discard the frame with PBB-TE MIP TLV. The loopback response from the MIP contains

- ESP-SA = GROUP MAC
- ESP-DA = Reverse MAC (or CBP-A's MAC)

Because MIP-1 is part of a point-to-multipoint ESP, it uses the SA as GROUP MAC.

Leaf Initiates Point-to-multipoint LBM

In the case of a leaf (CBP-C) initiating a loopback message, the PBB-TE MIP TLV contains

- Type = 9
- Length = 14
- Reverse VID = One from the ESP list of the TE-SID; used only by the MIP.
- MIP MAC Address = MIP MAC
- Reverse MAC = ESP-DA of the point-to-multipoint ESP, or GROUP MAC

The loopback response from the MIP contains

- ESP-DA = Reverse MAC (in this example, a GROUP MAC)
- ESP-SA = Reverse MAC field (in this example, a GROUP MAC)

Note: The ESP-DA is a GROUP MAC, because the only way to respond to CBP-C is through the point-to-multipoint ESP.

The payload should have ESP-SA in the received LBM as DA, so that the MEP which receives the LBM can validate it.

Linktrace Protocol in a PBB-TE MA

Linktrace Message

1. A Linktrace Message (LTM) is transmitted by a MEP associated with a PBB-TE MA. It uses as the destination_address parameter the MAC address indicated by the value of the ESP-MAC DA field of the MA's component ESP that has the MEP's MAC address indicated in its ESP-MAC SA field.
2. LTMs carry the PBB-TE MIP TLV constructed as follows:
   - The MIP MAC address field is null.
   - The Reverse VID field contains the parameter provided by an operator used as the vlan_identifier of the linktrace response (LTR).
   - The Reverse MAC field is included only if the MEP is associated with a point-to-multipoint TE service instance. If the MEP is a root MEP, the Reverse MAC field contains the ESP-MAC SA value of any of the point-to-point ESPs in the TE service instance. If the MEP is a leaf MEP, the Reverse MAC field contains the ESP-MAC DA of the point-to-multipoint ESP in TE service instance.
3. No special destination_address validation tests are performed by the ProcessLTM() procedure in the case of PBB-TE MAs.
4. The process to identify a possible egress port by an intermediate device that implements an MIP associated with a PBB-TE MA, queries the filtering database of the corresponding bridge, using the destination_address and the vlan_identifier of the LTM as the parameters for the lookup.

Linktrace Response

For a linktrace response issued by a PBB-TE MP

1. The destination_address is the value carried in the Reverse MAC field contained in the PBB-TE MIP TLV of the received LTM, if the ESP-MAC DA in the received LTM is a Group MAC address. Otherwise, it is the source_address of the received LTM.
2. The source_address is its MAC address if the LTR is issued by a PBB-TE MEP.

   In the case of a PBB TE MIP, the source address of LBR is the destination_address of the received LTM if it is an Individual MAC address. Otherwise, it is the value carried in the Reverse MAC field contained in the PBB-TE MIP TLV of the received LTM.
3. The vlan_identifier parameter is set to the value carried in the Reverse VID field contained in the PBB-TE MIP TLV of the received LTM.

LTM and LTR Frame Flow

Consider the figure below, which illustrates a point-to-multipoint TE service instance.

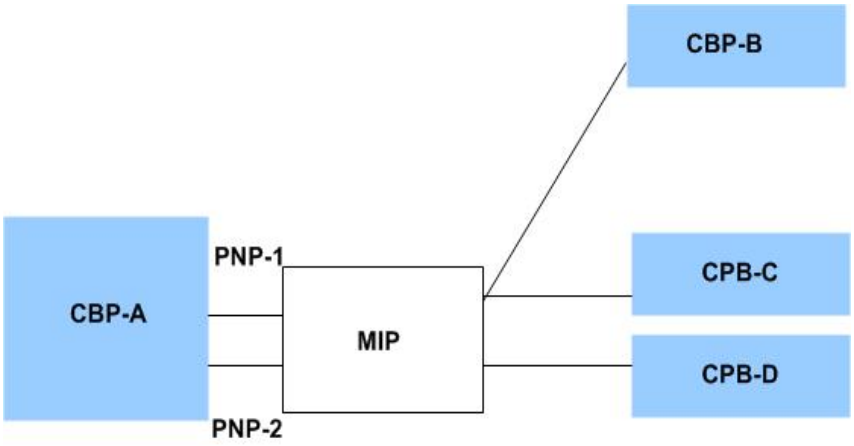


Figure 5-5: Point-to-Multipoint TE

In this scenario, when a multicast frame is initiated, the frame can reach CBP-D on the same point-to-multipoint ESP if PNP-1 and PNP-2 are configured with the same primary VID.

Point-to-Multipoint from Root

In the case illustrated, when the TE at CBP-A is associated with a point-to-multipoint TE-MA and initiates an LTM with the group MAC in the ESP-DA field, then the PBB-TE MEP transmits the PBB-TE MIP TLV with the following data:

- Type = 9
- Length = 14 (the Reverse MAC)
- MIP MAC address = NULL
- Reverse VID = One of the ESP-VIDs from the list of ESPs in the TE-SID
- Reverse MAC = ESP-SA of the point-to-multipoint ESP, in this example, ESP-SA of CBP-A

For an LTR from an MIP or any of the leaves:

- ESP-SA = The MEP's or MIP's MAC
- ESP-DA = SA field in the received LTM

Point-to-Multipoint from Leaf

When a node from CBP-C initiates an LTM with the Group MAC as the DA, then the PBB-TE MIP TLV contains:

- Type = 9
- Length = 14
- MIP MAC address = NULL
- Reverse VID = One of the ESP-VIDs from the list of ESPs in the TE-SID
- Reverse MAC = ESP-DA of the point-to-multipoint ESP, in this example, the group MAC address

For an LTR from a MEP

- ESP-SA = MEP's MAC
- ESP-DA = Group MAC

For an LTR from an MIP

- ESP-SA = Group MAC
- ESP-DA = Group MAC

Note: The ESP-DA is GROUP MAC, because the only way to respond to CBP-C from an MIP is to use a point-to-multipoint ESP.

In the payload, the DA field contains the ESP-MAC SA of CBP-C.

Addressing PBB-TE MIPs

PBB-TE MIPs are TE-instance based, but the MIPs are associated with a VID in a TESI and a level.

Evaluating PBB-TE MIPs

Managed objects control the creation of MIPs, but indirectly, rather than explicitly, as for MEPs. Every MA defined in a Bridge can cause the management entity to create MIPs on every Bridge Port. On each Bridge Port, the following conditions trigger the management entity to re-evaluate MIP creation permissions for a given VID (or list of VIDs):

- The bridge is initialized.
- a MEP is created or deleted on VID(s) *x* on that bridge port.
- An Up MEP is created or deleted on VID(s), which is not allocated to a TE service instance on any Bridge Port.
- A change occurs in whether VID(s) can or cannot pass through the Bridge Port.
- A change occurs in the Default MD Level managed object.
- A change occurs in a Maintenance Association managed object associated with VID(s).

Default MD Level Managed Object

The Default MD-Level managed object is maintained for each VID. When a PBB-TE MIP needs to be created, the TE VID and the level need to be identified by the operator.

Active Level

For each Bridge Port and TE service identifier *x,* which can be a list of VIDs, the Maintenance Entity creates a list of active MD Levels. This list contains:

- The MD Level of each of the MAs (if any) that monitors service *x*, and has a MEP configured on Bridge Port *p*
- The MD Level of each of the MAs (if any) that monitors service *x*, and has at least one Up MEP configured on some Bridge Port in this Bridge other than Port *p*
- The MD Level of each of the MAs (if any) that monitors te-instance *x*, and has no MEPs configured on any Bridge Port in this Bridge
- The MD Level of the entry in the Default MD Level managed object for service *x*

There is exactly one MD Level *d* in the active list that is eligible for MIP creation. It is the lowest MD Level in the list of active MD Levels such that there is no MEP configured on Bridge Port *p* and VIDservice *x* at MD Level *d* or at any higher MD Level. This is the lowest MD Level in the list, if there is no such MEP configured. The Maintenance Entity then uses Table 22-2 to determine whether the MIP is to be created.

The enumerations defined above control the creation of MIPs for a VID identified by parameter *x* on a Bridge Port. They specify permissions for MIP creation given by MA, MD and Default MD Level object.

- **defMHFnone** No MHFs (MIP Half Functions) can be created for this VID(s).
- **defMHFdefault** MHFs can be created for this VID(s) on any Bridge Port through which the VID(s) can pass where:
    - There are no lower active MD levels, or

  - There is a MEP with the TE-SID (to which this VID is associated) at the next lower active MD-level on the port.
- **defMHFexplicit** MHFs can be created for this VID(s) only on Bridge Ports through which this VID(s) can pass, and only if there is a MEP at the next lower active MD level on the port.
- **defMHFdefer** In a Maintenance Association managed object only, the control of MHF creation is deferred to the corresponding variable in the enclosing Maintenance Domain.

PBB-TE MIP TLV Format

There is an additional PBB-TE MIP TLV for Loopback Message/Loopback Response (LBM/LBR) and Linktrace Message/Linktrace Response (LTM/ LTR) in a PBB-TE MA. This PBB-TE MIP TLV must be included in every LTM transmitted by a MEP configured on a PBB-TE MA, and must be included in every LBM transmitted by a MEP to a MIP configured on a PBB-TE MA. This TLV need not be present in any other type of MA.

Type = 9 This value is used only in MAs monitoring PBB-TE services. It enables intermediate MIPs to selectively intercept LBMs that are targeting them. The PBB-TE MIP TLV is not used if the target address of the LBM or LTM is associated with any of the values in the ESP-MAC DA field of the monitored MA's component ESPs. Otherwise, it shall be present.

Length When there is no Reverse MAC field present, the Length field is 8, otherwise it is 14, identifying the Reverse MAC field

MIP MAC address The MIP MAC address contains the destination MAC address for LBMs transmitted by the MEP. In the case of LTMs, this field in null.

Reverse VID The Reverse VID field is used by MIPs in order to set vlan_identifier parameters in LBRs and LTRs. Its value is supplied by the operator during the initiation of an LBM or an LTM. In the case of point-to-point TESI, the value is the ESP-VID of the MA's component ESP that has the MEP's MAC address in its ESP-MAC DA field.

Reverse MAC When LBM or LTM is initiated on a root PBB-TE MEP, the Reverse MAC field refers to the ESP-MAC SA of any of the point-to-point component ESPs. When LBM or LTM is initiated on any of the leaf MEPs, then the Reverse MAC field refers to the ESP-MAC DA of the point-to-multipoint ESP (which is a group MAC).

Mismatch Detection

A Traffic-Bit field in PBB-TE Connectivity Check Messages addresses issues with mismatch defects.

MEP Mismatch Variables

The following variables are local to the MEP Mismatch state machines for a PBB-TE MEP implementing the Traffic field:

- mmCCMreceived
- mmCCMdefect
- mmCCMTime
- disableLocdefect
- mmLocdefect

mmCCMreceived A Boolean flag set to true when rcvdTrafficBit does not match the presentTraffic.

mmCCMdefect A Boolean flag set and cleared by the MEP Mismatch state machines to indicate that one or more CCMs with Traffic fields not matching the presentTraffic have been received, over a period that is 3.5 times the configured CCM transmission rate.

mmCCMTime The time used to initiate the mmCCMwhile timer. This is the time taken for a remote BEB (backbone edge bridge) to send a CCM with the correct information in the Traffic field. It is tied to the target protection switching time, or 50ms, as a mismatch is normal prior to the completion of a protection switch.

In the simplest case, mmCCMTime is equal to the Detection Time + Restoration Procedure Time + Restoration Transfer time. This gives a time of (3.5 * CCM interval + Restoration Procedure Time + 1.25 * CCM interval). As a result, the value is set to a maximum 50ms, or (4.75 * CCMtime (CCMinterval) + 10ms).

disableLocdefect A Boolean that disables the indication of the mmLocdefect. The variable is always set to 1 for PBB-TE MAs that are not part of a TE protection group. On a PBB-TE MEP associated with the working entity in one or more TE protection groups to which its monitoring TESI MA is assigned, disableLocdefect is true if and only if either p.SF (protection Switch Failure) or LoP (Loss of Protection) is true on any of these TE protection groups. On a PBB-TE MEP associated with only one protection entity in all the TE protection groups to which it is assigned, disableLocdefect is true if and only if to FS (Force Switch) is true is true on this TE protection group.

mmLocdefect A Boolean flag set and cleared by the MEP Mismatch state machines to indicate that presentmmLoc is set to 1, over a period of 50ms.

MEP Mismatch State Machines

The MEP Mismatch state machines implement the functions depicted in the state diagrams below, and the MEP Mismatch Variables defined above. There is one MEP Traffic Field Mismatch state machine and one MEP Local Mismatch state machine per PBB-TE MEP that implements the Traffic field.

MEP Traffic Field Mismatch State Machine

This state machine is run when a CCM frame is received and there is a mismatch between the present traffic and the received traffic bit.

MEP Local Mismatch State Machine

This state machine is run when a CCM frame is received, a defect is present, and Automated Protection Switching (APS) in PBB-TE is initiated leading to protection switching.

PBB-TE Protection Switching

In contrast to PBB Spanning Tree Protocols, broadcasting and flooding are not used in PBB-TE. Filtering databases are populated using a network management system or control plane, allowing Ethernet Switched Paths (ESP) to be engineered and provisioned across a network. PBB-TE provides end-to-end linear protection for point-to-point TE service instances (TESI), where a dedicated protection point-to-point TESI is established for a specific point-to-point TESI, and traffic is automatically switched from a working TESI to a protection TESI when a failure occurs on the working entity. The protection entity is a pre-established entity that enables the availability of resources when a defect is detected and a corresponding sub-50ms switchover takes place.

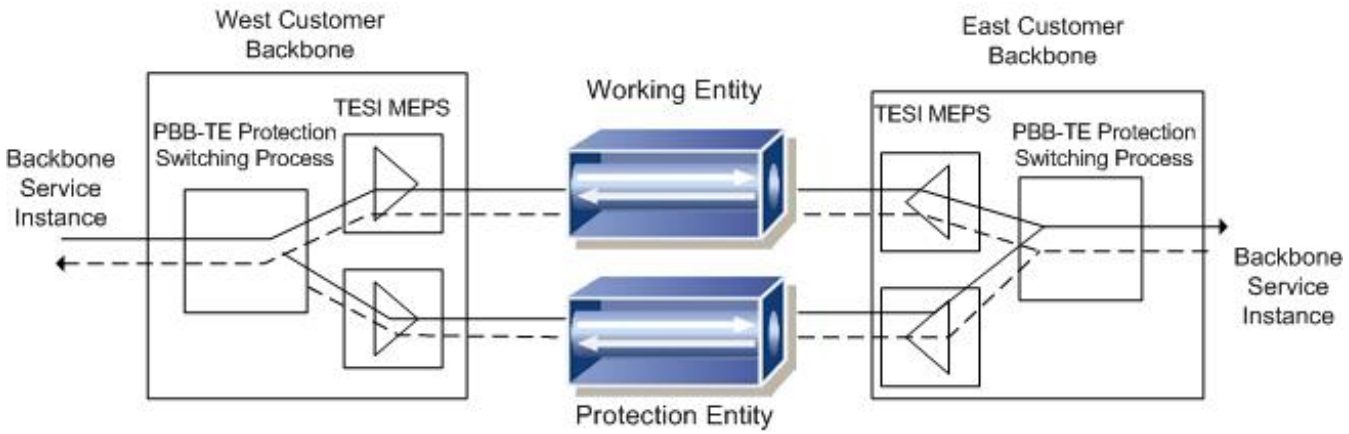


Figure 5-6: PBB-TE Protection Switching Topology

IEEE 802.1Qay specifies linear point-to-point 1:1 bi-directional and load-sharing architectures. The 1:1 bi-directional mode can be revertive and non-revertive. In the 1:1 architecture, traffic is sent onto only one path at any point in time, based on information from Operations, Administration and Management (OAM) processes, protection switching processes, or network operators. Protection Switching may be triggered by manual operation or by Connectivity Fault Management (CFM) information arising during periodic monitoring of the working and protection paths, or from physical layer monitoring, such as loss of signal or other defects detected through Connectivity Fault Management (CFM).

The starting point in providing protection switching is the creation and configuration of a TE protection group. A TE protection group is a set of two point-to-point TE service instances between a pair of Customer Backbone Ports (CBPs), which carries an assigned set of Backbone Service Instances (BSI), and continues to carry the BSIs if any of the TESI in the protection group fails or is disabled. Because there is no control plane protocol that carries Protection Switching information from end-to-end, both ends of protection switching are dependent upon local information, for example, CFM or Operator Requests. Protection Switching makes use of Connectivity Check Messages (CCMs) as defined in IEEE 802.1ag. CCM is the standard Ethernet mechanism for detecting and signaling connectivity failures.

Protection Switching requires that both the working and protection trunks are monitored, and this is accomplished with CCM. Each PBB-TE trunk is monitored by an independent Maintenance Association (MA), so that one MA is set to monitor the Working PBB-TE trunk while another PBB-TE MA is set to monitor the Protection PBB-TE trunk.

1:1 Bidirectional Protection Switching Mode

In the 1:1 bi-directional mode, one trunk is dedicated as a working trunk, and no data traffic flows on a protecting trunk until a protection switch takes place.

Load Sharing Protection Switching Mode

The protection switching mechanism is capable of load sharing, because the TESI assigned to a TE protection group can be reused in a number of TE protection groups, thereby enabling a list of I-SIDs to be distributed among a set of interdependent TE protection groups. A set of interdependent protection groups comprise a coordinated protection group. Theoretically, a B-Component is capable of translating incoming I-SID values by comparing the Local-SID stored in the BSI table to a Customer Backbone Port. A BSI table is configured by an operator, and has one entry for each BSI supported on a CBP. The BSI table is used to filter frames containing an I-SID value that is not configured on the CBP. It can also be used to enable BSI-specific assignments of a B-VID, to translate I-SID values, or to translate B-DA. The BSI table should contain one or more of the following fields for each entry:

- B-SID = Backbone Service Instance ID
- B-VID = Backbone VLAN ID

- Local-SID This field is used to perform a bi-directional 1:1 mapping between BSID and Local-SID.
- Default Backbone destination

Examples

Suppose that TE-SID2 belongs to TE-Protection-Group1 and TE-Protection-Group2, and a system failure occurs on TE-SID2. In this event, in:

Group 1

- TE-SID2 is the working TESI
- TE-SID1 is the protection TESI

and in:

Group 2

- TE-SID3 is the working TESI
- TE-SID2 is the protection TESI

If TE-SID2 subsequently fails, TE-Protection-Group1 engages protection switching and moves its ISID to TE-SID1, in which case TE-Protection-Group2 loses its protection path.

To illustrate a more common case:

In Group 1

- TE-SID1 is the working TESI
- TE-SID2 is the protection TESI

and:

In Group 2

- TE-SID3 is the working TESI
- TE-SID2 is the protection TESI

In this scenario, if TE-SID1 fails, TE-Protection-Group1 engages protection switching and moves its ISIDs to TE-SID2. In the same manner, if TE-SID3 fails, TE-Protection-Group2 also engages protection switching and moves its ISIDs to TE-SID2.

Both protection groups share the protection path (TE-SID2) that is meant to be an occasional backup or fallback path.

Note: A TE-SID is unique system-wide (per-BEB), so the same TE-SID on two different CBP is not possible. Errors are detected by Connectivity Fault Management in PBB-TE and Automated Protection Switching (APS).

Mismatch Detection

Due to equipment malfunctions or incorrect configurations, a mismatch between the mappings of the backbone service instance to appropriate TE service instance at terminating CBPs can take place. To maintain the proper operation of the network, these mismatches need to be detected and reported.

There are two types of mismatches:

1. Protection Switching Incomplete
2. Working/Protection configuration mismatch

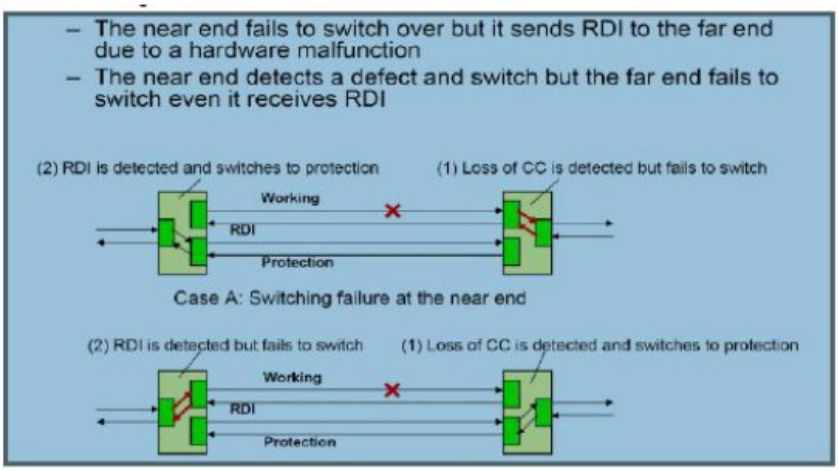


Figure 5-7: Mismatch Detection

In this example, one end is configured to send traffic on the working entity while the other is configured to send traffic on the protection TESI. One end is configured in revertive mode while the other end is configured in non-revertive mode. PBB-TE maintenance end points (MEPs) that support the traffic bit field can detect this defect. The mismatch state machines support this functionality.

CFM In Protection Switching

Because every TESI can be uniquely monitored by CFM, as soon as a defect is detected, CFM initiates protection switching. Protection switching is supported by these processes:

- An MA is created for each trunk in the trunk protection group, one for the working trunk and another for the protection trunk.
- Two UP MEP are configured for each MA, one MEP on each CBP on each edge trunk.
- Unicast CCMs are generated by MEPs on the trunk and are sent out.

Note: PBB-TE 1:1 bidirectional point-to-point Protection Switching CC messages must be unicast, while point-to-multipoint TESI can support multicast CCM, but not protection switching.

CFM and PBB-TE PS Interactions

In order to achieve Protection Switching, PBB-TE PS depends upon Connectivity Fault Management. The following interactions take place between CFM and PBB-TE APS:

- When CCM LOC is detected, CFM signals PBB-TE-PS and traffic is switched to the protection TESI CFM is informed. This is required for CFM for TrafficBit and PresentTraffic Bit fields.
- Upon receiving RDI from the far end, CFM informs local protection switching, which makes changes in the path.
- When there are any mismatches, detection of mismatches takes place with the used of the TrafficBit field in CCM. When mismatches are detected, CFM initiates Protection Switching.

PBB-TE Protection Switching Design

Protection Switching in PBB-TE is driven by state machines that receive local events as input. Most local events are sent from CFM processes. For example, CFM detects a failure condition and generates an event, which is used as inputs to the PBB-TE-PS state machine. Possible events that occur that cause transition to a new state include:

Priority	Request
Highest	LoP = Loss of Protection
	FS = Force Switch
	p.SF = Protection Switch Failure
	w.SF = Working Switch Failure
	MStoProtection = Manual Switch to Protection
	MStoWorking = Manual Switch to Working
	WTR = Wait to Restore
Lowest	No Request

Since all protection switching is driven by CFM, PBB-TE PS functionality is implemented in the ZebOS-XP ONMD module. When Protection Switching requires, CFM signals ONMD, and ONMD in turn signals NSM to map traffic to protecting TE service instances.

PBB-TE MEP Placement in Bridge Port

PBB-TE MEPs are always Up MEPs, and may only be placed on CBPs because they correspond to the demarcation points of the Ethernet Switched Paths (ESP) supporting the PBB-TE service. In principle, for each TE service instance there can be zero to eight Up MEPs, ordered by increasing Maintenance Domain (MD) level from frame filtering to port filtering, although not more than one MD level is expected for PBB-TE Maintenance Associations (MA). PBB-TE MEPs may do the following:

1. Set the traffic field bit on transmitted CCMs based on information from the BSI table associated with the CBP on which the MEP is configured. When the traffic field bit is set, so is the presentTraffic variable.
   - **presentTraffic** A Boolean value indicating whether at least one Backbone Service Instance is configured to use the TESI's ESP upon which this PBB-TE MEP is transmitting CCMs. PresentTraffic field is true if the Backbone Service Instance table of the CBP associated with this MEP contains an entry that has in it's B-VID and Default Backbone Destination fields the values of ESP-VID and ESP-DA of the TESI's ESP, which originates at the MEP.
   - **Traffic Field** In the case of a PBB-TE MA, the second most significant bit of the Flags field is the Traffic bit. This bit, if supported, is used to indicate the Backbone Service Instances in the monitored TE service instance. This bit is set to 1 if the transmitting MEP's present Traffic variable is set and 0 if not. This bit is only examined by PBB-TE MEPs.
2. Check the traffic field bit of CCMs received.
3. Implement the MEP Mismatch State Machine.

Protection Switching Process

As soon as a defect is detected in the working entity, traffic flowing on the working path must be switched to the protected path. This is done by setting the VID values of the corresponding I-SID entry in the BSI Table to the ESP-VID (*p*) or P-VID parameter (when the B-VID column in the CBP's BSI table is not supported). If B-VLAN identifier is not supported, Protection Switching with load sharing is not possible. This means that the BSI table at a CBP needs to be updated with corresponding ESP-VID.

CCM Detection and APS Operation

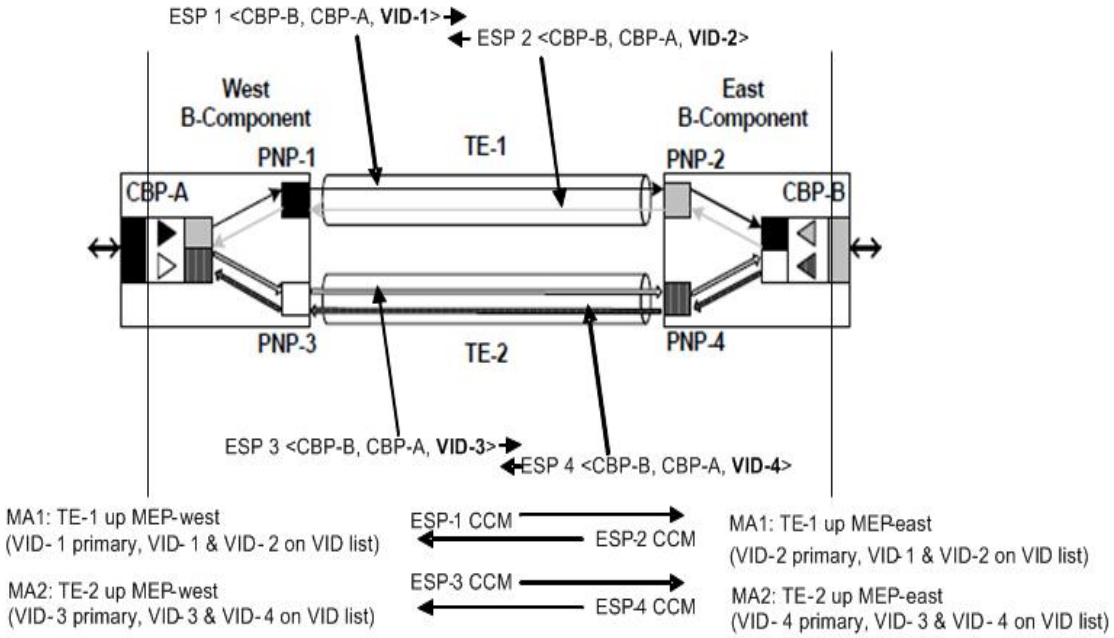


Figure 5-8: CCM Detection and APS Operation

CFM for point-to-point 1:1 linear protection requires two MAs, one for each ESP service instance. They are illustrated by TE-1 and TE-2 in the diagram above. Each MA in turn has two VIDs and maintains two up MEPs, one at each end of the ESP CBP endpoint. a MEP can only send CFM frames using its primary VID, but can receive CFM frames on any VID that is on its VID list. In the illustration above, if ESP-1 fails, TE-1 up MEP-East detects the CCM Loss Of Continuity (LOC) and switches CBP-B ISID to BVID mapping from VID-2 (shaded in gray, ESP-2) to VID-4 (shaded in gray, ESP-4). TE-2 Up MEP-East continues to send CCM frames on ESP-4 with Remote Defect Identification (RDI) set for as long as it does not receive valid CCM frames. Upon receiving a CCM frame with RDI set, a MEP causes its associated CBP to switch the I-SID to BVID mapping. In this example, TE-2 Up MEP-West receives CCM frames with RDI set, which causes CBP-A to switch its I-SID to BVID mapping from VID-1 (shaded black, ESP-1) to VID-3 (shaded black, ESP-3) thus completing the APS failover procedure.

PBB-TE Protection Switching State Machines

PBB-TE PS state machines include the following:

- Hold-Off-State Machine
- Clear Manual Switch State Machine
- Service Mapping State Machine

The relationships between the PS state machines are illustrated in The following diagram. Open arrows indicate variables that are set by one state machine, and both read and set by another. Closed arrows indicate variables set by the owning state machine, and only read by another.

PBB-TE PS logic is sub-divided into three state machines, and each function is implemented independently.

fsm_hold_off_func

The fsm_hold_off_func is only called when a working switch failure (w.SF) or protection switch failure (p.SF) takes place. This function only has rights to update the pSFH or wSFH flags in protection group structure.

fsm_clear_manual_switch_func

The fsm_clear_manual_func is only called upon clearing of an administratively configured state, for example, manual switch to protection or manual switch to working.

fsm_service_mapping_func

This FSM contains the main logic for mapping the traffic to working or protection service instances. The design of this FSM is based upon incoming events. Execution of all incoming events is based upon the current state the FSM is in. In order to simplify the implementation, a separate function is mapped to each possible state.

Hold Off State Machine

The Hold off state machine needs to be present only when Hold-Off timer is supported. The suggested range for the hold-off timer is <0-10> seconds in steps of 100ms. The default value is 0. If the hold-off timer is not configured, its value is 0, thus there are no state changes related to hold-off timing. The Hold-Off timer is only applicable when a p.SF (protection switch failure) or w.SF (working switch failure) event takes place.

Clear Manual State Machine

The clear manual state machine is executed when any administrative command is given to clear an existing Manual Switch. This state machine is executed when clearing the MSTOWorking or MSTOProtection administrative commands. When the MSTOWorking command is issued, control reaches to the Service Mapping State Machine.

Transitions for Service Mapping State Machine

1:1 Bi-directional Revertive/Non-revertive Local State

	State	LOP	FS	pSF	wSF	MSTO Working	MSTO Protection	WTR	Clear
A	No Request Working/Active Protection/Standby	→H	→G	O	→E	→D	→C	N/A	O
B	WTR Working/Standby Protection/Active	→H	→G	→A	→E	→D	→C	→A	→A
C	MSTOProtection Working/Standby Protection/Active	→H	→G	→A	→E	O	O	O	→A
D	MSTOWorking Working/Active Protection/Standby	→H	G	→A	→E	O	O	O	→A
E	wSF Working/Standby Protection/Active	→H	→G	→A	O	O	O	O	O
F	FS Working/Standby Protection/Active	→H	O	O	O	O	O	O	→A
G	LOP Working/Active Protection/Standby	O	O	O	O	O	O	O	→A

Mismatch Handling

When mismatches are detected in incoming CCM, the end that detected the mismatch waits for a period of time, then informs the Protection Switching process if the mismatch still exists. In this situation, APS treats the event as a signal failure (SF).

PBB-TE-APS Load Sharing

ZebOS-XP supports Provider Backbone Bridge - Traffic Engineering (PBB-TE) APS (Automatic Protection Switching) load sharing as specified in the IEEE 802.1Qay requirements.

PBB-TE APS Load Sharing Features

ZebOS-XP PBB-TE APS load sharing supports the following features:

- Identifies the protection switching mechanisms for the protection groups that are part of load sharing
- Buffers CCM transmission data for every outgoing CCM, This feature avoids creating a CCM message for all transmissions, even if there is no change in a message. This also improves the scalability of transmission of CCM frames and reduces the CCM transmission interval.

PBB-TE Description

ZebOS-XP PBB-TE APS load sharing provides an end-to-end; one-to-one bidirectional linear protection switching that is capable of load sharing for point-to-point TE Service Instances (TESIs) within a PBB-TE region. In a PBB-TE region, an IB-Backbone Edge Bridge (IB-BEBs) is the demarcation point between the Provider Backbone Bridged Network (PBBN) and all attached networks. The protected domain is the area between each Customer Backbone Port (CBP) on the different components of the configured IB-BEBs.

There are ISIDs (I-component Service IDs) mapped for each TE-Protection group. When protection switching occurs, the ISIDs maps to the egress backbone VLAN ID (B-VID) of the TE service instance, which is active. When there are TE Service instances that are re-used in a number of TE protection groups, the load on the ISIDs mapped to the TE protection groups are shared across a set of TESIs and are protected against single faults by 1:1 protection.

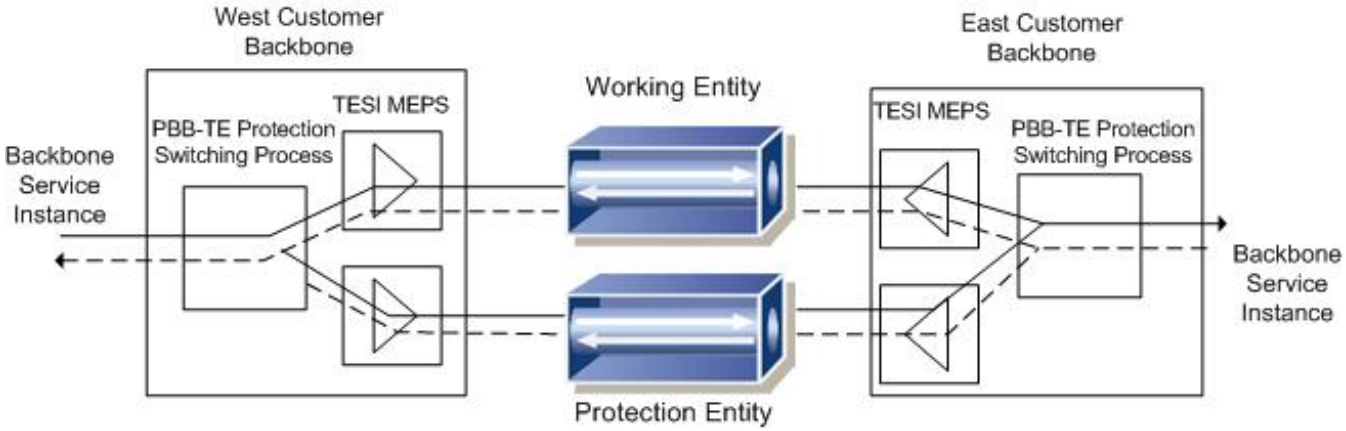


Figure 5-9: Simple PBB-TE Example

Architecture

PBB-TE APS load sharing allows a single TESI to be a TESI member entity in multiple protection groups. In the example below, two protection groups share one TESI with the two other protection groups a part of the load sharing as working paths. The TESI 2 tunnel shares the two protection groups. Figure 5-10 displays a high-level PBB-TE APS load sharing architecture.

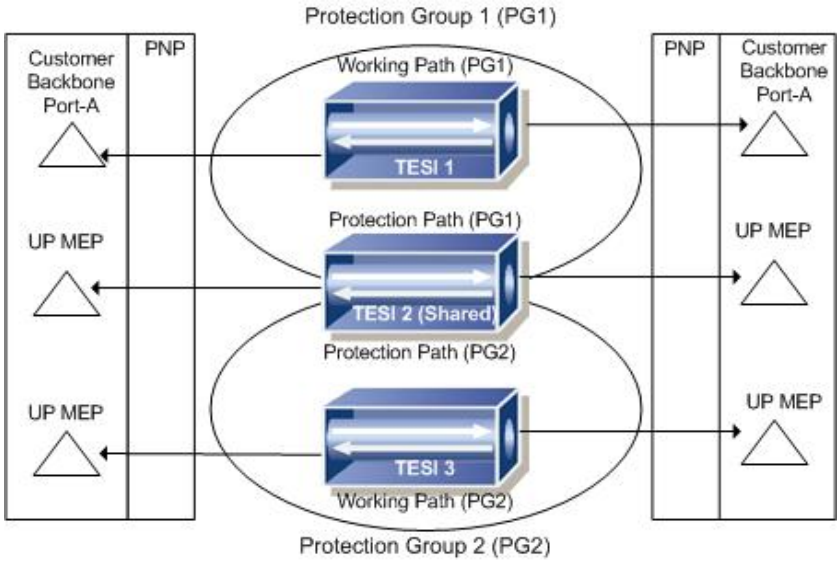


Figure 5-10: High-level PBB-TE APS Load Sharing Architecture

The switching mechanism on the load shared TESIs is according to the TESI role in each protection group. For example, to remove traffic unconditionally from a TESI, a request to lock the TESI results in a state change request to each of its TE protection groups. The state change request is either LoP (Loss of Protection) or FS (Fail Stop) according to its role in the protection groups. Similarly, a request to switch manually the traffic from a TESI results in a state change request that is either Manual Switch to Working or Manual Switch to Protection, which is also based on the role of the TESI in each protection group. Figure 5-11 displays the TESI load sharing switching mechanism.

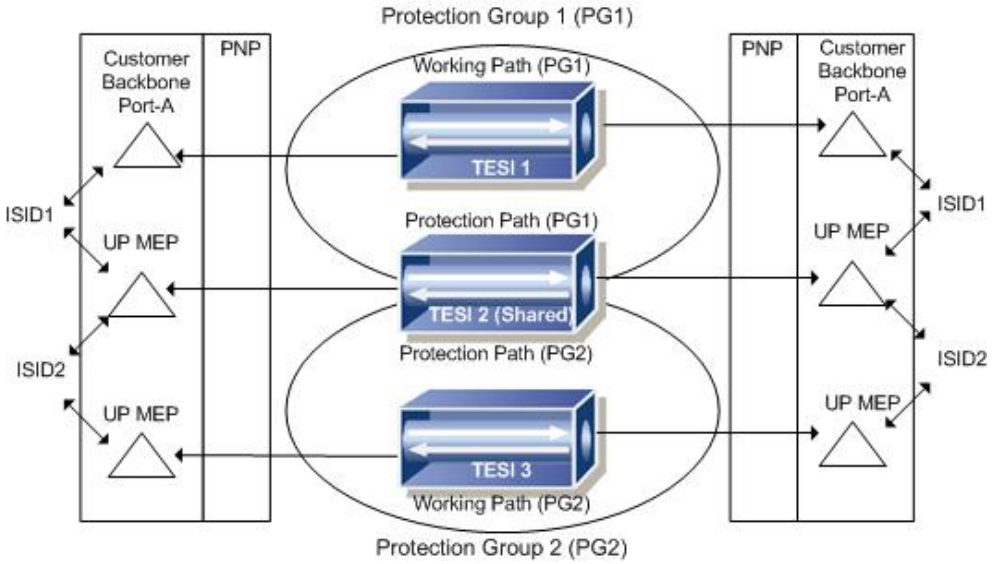


Figure 5-11: PBB-TE APS Load Sharing Switching Mechanism

In the above diagram:

1. ISID1 maps to PG 1 and ISID 2 maps to PG 2.
2. For PG 1, TESI 1 is a working path and TESI 2 is the protection path.
3. For PG 2, TESI 3 is a working path and TESI 2 is the protection path.
4. By default, each ISID maps to a working-path egress Backbone VLAN ID (B-VID), so ISID 1 flows through TESI 1 tunnel and ISID 2 flows through TESI 2 tunnel.
5. If there is a failure on TESI 1, the traffic from ISID1 automatically flows through the TESI 2 tunnel.
6. If there is a failure on TESI 2, the traffic from ISID 1 maps to TESI 1 egress B-VID and the traffic from ISID 2 maps to egress B-VID of TESI 3. In this way, three TESIs share the list of ISIDs that map to the two TE protection groups. Load sharing is achieved in PBB-TE APS by reusing the TESIs in a number of TE protection groups.

Switch Over

When the TESI is part of multiple protection groups, the TE protection group that is part of different groups does the switching and changes the state as per the working/protection state of the group. Switching can occur because of one the following scenarios:

1. Interface goes down (see Interface is Down on page 160)
2. CCM loss of connection or disabled (see When CCM Loses Connectivity or is Disabled on page 161)
3. User administratively locks a TESI (see User Administratively Locks a TESI on page 161)
4. User administratively does a manual switch on a TESI (see User Administratively does a Manual Switch on a TESI on page 162)

Interface is Down

By default, the paths of each TESI are UP. When a link goes down on the egress PNP port of a TESI in a customer backbone port (for example, CBP-A), a link down message from the NSM triggers a Working Signal Failure (W-SF), which is sent to the remote end by a protection TESI using APS signaling. This triggers a W-SF event at the other customer backbone port (for example, CBP-B). When an interface goes down on the shared TESI on the egress PNP at the first customer backbone port (for example, CBP-A):

1. A local Protection Signal Failure (P-SF) event triggers for the first protection group (PG1) at CBP-A.
2. A local W-SF event triggers for the second protection group (PG2) at CBP-B.
3. Now, if the PG1 protection path is down at CBP-B, then there is no APS signaling and the RMEP down event from CFM triggers P-SF for PG1. This is down after the RMEP timeout interval.
4. The event is W-SF in PG2. Since protection path is active, the W-SF event is sent to CBP-B by APS signaling on the protection path of another TESI.

When the egress PNP of a TESI at CBP-A is down:

1. CBA-A receives a link down message and the two protection groups (that is, PG1 and PG2) triggers by the event.
2. This event cannot be sent to the remote end by APS signaling as the protection path is down.
3. The RMEP down message on a TESI from connectivity fault management (CFM) triggers P-SF on both protection groups of CBP-B.

When the egress PNP of a TESI is down at the first customer backbone port (for example, CBP-A):

1. A link down message triggers a W-SF on the first protection group (PG1) of CBP-A.

2. The shared TESI receives a W-SF event, as this TESI provides protection for both the protection groups (that is, PG1 and PG2).
3. A W-SF event triggers after it receives the RMEP down message on the shared TESI.
4. A W-SF event is ignored when the W-SF event is received on a protection path that protects more than one PG.

When CCM Loses Connectivity or is Disabled

Scenario One

By default, the paths of each TESI are UP.

1. When CCMs are disabled on the first CBP (that is, CBP-A) of a TESI, the RMEP state machine at the second CBP (that is, CBP-B) receives a message stating that the TESI on CBP-A is down.
2. This triggers a W-SF event at the second CBP.
3. In addition, the second CBP sends the CCMs a Remote Defect Indication (RDI) bit set until the MEP of the TESI on the first CBP is up.
4. This working signal failure causes switching from the TESI on first protection group (PG1) to the TESI on the second protection group (PG2) at CBP-A.

Scenario Two

By default, the paths of each TESI are UP.

1. When CCMs are disabled on the shared TESI at CBP-A, an RMEP down message from the CFM at the second CBP (that is, CBP-B) triggers P-SF on the first protection group (that is, PG1) and a W-SF on the second protection group (that is, PG2).
2. When the CBP-A receives the CCMs with RDI bit set for TESI 4 MEP, the switching at CBP-A happens accordingly.

User Administratively Locks a TESI

When a user administratively locks a TESI and if the TESI is part of multiple protections groups, then a loss of protection (LoP) or Fail Stop (FS) passes to either the protective TESI or a working TESI, respectively. The LoP on a TESI locks out the protection path if the TESI is a protective TESI in that protection group, and the FS does a force switching to the protected TESI if the TESI locked is a working TESI in that protection group.

The state machines and the states are per protection group, so depending on the state change, far end requests are signaled, accordingly.

Scenario One

Assuming that a lock is set on the shared TESI on the first CBP (that is, CBP-A):

1. If this TESI is protective in the first protection group (that is, PG 1), the state changes to LoP and a message is sent to the shared TESI 4 on the second CBP (that is, CBP-B).
2. Upon receiving the frames at CBP-B, the system loops through the protection groups on which the TESI is a protective TESI.
3. Next, it finds the first protected group (that is, PG 1) and switching occurs.
4. Finally, the state changes to lockout of protection (LoP) at CBP-B.

If the shared, TESI is working in PG2, the state change is FS. This information is sent on a protective TESI of the protection group. In addition, the state change of PG2 is sent to the other TESI. When the other TESI at the CBP-B receives the state change APS signaling the state changes to FS.

Scenario Two

Assuming that a lock is set on a TESI of CBP-A:

1. If the TESI of CBP-A is a protective TESI in both the protection groups, then the change state is LoP in both TE protection groups.
2. APS signaling sends a message regarding the state change to LoP to the far end CBP-B.
3. This triggers the state change to LoP in both protection groups.

User Administratively does a Manual Switch on a TESI

When a user administratively issues a manual switch on a TESI and if the TESI is part of multiple protection groups, then either a Manual Switch to Protection or Manual Switch to Working event is issued to the working TESI or protective TESI, respectively. The state machines and the states are per protection group, so depending on the state change, far end requests are signaled, accordingly.

Scenario One

Assuming that a manual switch is given on a shared TESI of CBP-A:

1. If the shared TESI is protective in the first protection group, then the state changes to Manual Switching to Working. A message is sent to the shared TESI on CBP-B.
2. Upon receiving the frames at CBP B, the system loops through the protection groups on which the TESI is a protective TESI.
3. If it finds PG1, then switching occurs.
4. Finally, the state changes to manual switch to working at CBP-B.

Scenario Two

1. If the shared TESI is working in PG 2, then the state change is manual switching to protection.
2. If this information is on the protective TESI of the PG, then the state change of PG2 is sent to another TESI.
3. Upon receiving the state change APS signaling at the TESI of CBP-B, the state changes to FS.

Scenario Three

Assuming that manual switch is given on a TESI of CBP-A:

1. If the TESI of CBP-A is a protective TESI in both the protection groups, the state changes to Manual Switch to Working in both protection groups.
2. APS signaling sends a message that the state is now manual switch to working to the far end of CBP-B.
3. This triggers a state change to manual switch to working in both protection groups.

System Design

The following section describes the system design for PBB-TE—APS Load Sharing.

APS Signaling Reception

The APS Automatic Protection Switching module is responsible for signaling reception changes. An APS frame is one of the OAM frames that is received through CFM socket. Upon receiving an APS frame on a TESI, if the TESI is part of load sharing, then the signal received is considered for each protection group by looping through the protection group.

APS Manual Switching

APS manual switching is part of multiple protection groups for a TESI, which is part of load sharing. A manual switch command on a TESI performs a manual switch to working or protection based on the TESI in a specific protection group.

Load sharing TESI switching per TE-Group

Coordination TE protection group switching is per TESI. That is, there are administrative commands for manual switching, loss of connection and, and fail start, which can be issued on a per TESI basis. For example, if an FS is issued on a protection group, a TESI is active for that PG-1. In addition, the state change is sent to the far end. The far end request is received on a protective TESI and PG is not there in the APS signaling. Therefore, when a far end FS request is received on a protected TESI that belongs to two protection groups, the switching occurs in both PGs at the far end. However, for protection groups that are part of load sharing, administrative switching is per TE group.

Lesser CCM Transmission Interval

To achieve lower CCM Transmission interval, the CCM transmission data is buffered for each outgoing CCM. This avoids sending a CCM message for every transmission, even if there is no change in the message.

Associating Protection Groups

Associating protection groups to an ISID is done at Network System Module (NSM). A PG create message is sent to ONMD. At ONMD, MA and MEP are created for each TESI. CCM convergence checks for each TESI. A protection group is associated to the working MA and protection MA. When there is a state change detected at one side of the protection group, the state change APS signaling is sent to the far end. When APS signaling is received, the far end state machines changes based on the state change received. If there is a switching, then the Protection Group ID and current active TESI ID is sent to NSM. At NSM, the ISID associated to the protection group maps to the egress ESP-BVID of the active TESID.

Data Structures

The functions in this chapter refer to the data structures described in this section.

See Chapter 2, *Connectivity Fault Management* for these data structures:

- cfm_1dm_rx
- cfm_dmm
- cfm_lb
- cfm_lm
- cfm_md
- cfm_tput_reception

cfm_lt

This structure in `onmd/cfm/cfm_ltm.h` defines the CFM link trace.

Member	Description
mep	Pointer to MEP if the LT is linked to a MEP
mip	Pointer to MIP if the LT is linked to a MEP
next_ltm_transid	Transaction ID for the next LTM; used by LT Initiator
curr_ltm	The current LTM Message from this MEP
ltm_reply_queue	For Link Trace Responder, list of pending LTRs
npending_ltrs	For Link Trace Responder, number of pending LTRs
ltf_while	For Link Trace Responder, random timer started to send LTR
rcvd_ltm	Received LTM messages
total_ltrs_tx	Total number of LTR transmitted
tx_ltm_result_ok	True if mep is active, otherwise false
active	Whether active
cli	CLI command
ltr_src_addr	LTR source address
ltm_pbb_te_mip_tlv	PBB-TE MIP TLV as per clause 21.7.5 in IEEE-802.1Qay-d3-5

```
struct cfm_lt
{
  /* Pointer to MEP if the LT is linked to a MEP */
  struct cfm_mep *mep;
  /* Pointer to MIP if the LT is linked to a MEP */
  struct cfm_mip *mip;
  /* Transaction ID for the next LTM. Used by LT Initiator */
  u_int32_t next_ltm_transid;
  /* The current LTM Message from this MEP */
  struct cfm_ltm *curr_ltm;
  /* For Link Trace Responder, list of pending LTRs */
  struct cfm_ltm_reply ltm_reply_queue;
  /* For Link Trace Responder, number of pending LTRs */
  u_int32_t npending_ltrs;
  /* For Link Trace Responder, Random timer started to send LTR */
  struct thread *ltf_while;
  u_char rcvd_ltm;
#define CFM_LTM_RECEIVED                1
  u_int32_t total_ltrs_tx;
  /* As of now this is true if mep is active else false */
  bool_t tx_ltm_result_ok;
  u_char active;
  struct cli *cli;
```

```
  /* PBB-TE MIP TLV as per the clause 21.7.5 in IEEE-802.1Qay-d3-5 */
#if defined HAVE_CFM && defined HAVE_PBB_TE
  u_int8_t ltr_src_addr[ETHER_ADDR_LEN];
  struct cfm_pbb_te_mip_tlv ltm_pbb_te_mip_tlv;
#endif /* HAVE_CFM && HAVE_PBB_TE*/
};
```

Command API

The functions in this section are called by the commands in the *Carrier Ethernet Command Reference.*

Function	Description
cfm_pbb_te_1dm_rx_enable	**Enables one-way delay measurement reception for a MEP**
cfm_pbb_te_add_ma	Adds a maintenance association
cfm_pbb_te_add_md	Adds a maintenance domain
cfm_pbb_te_add_mep	Adds a maintenance end point
cfm_pbb_te_add_rmep	Adds a remote maintenance end point
cfm_pbb_te_ais_disable	Disables Alarm Indicator Signaling
cfm_pbb_te_ais_enable	Enables alarm indicator signaling
cfm_pbb_te_ais_set_tx_interval	Sets the transmission interval for alarm indicator signaling
cfm_pbb_te_cc_mcast_disable	Disables multicast continuity check messages
cfm_pbb_te_cc_mcast_enable	Enables multicast continuity check messages
cfm_pbb_te_cc_unicast_disable	Disables unicast continuity check messages
cfm_pbb_te_cc_unicast_enable	Enables unicast continuity check messages
cfm_pbb_te_exm_send	Sends experimental OAM frames
cfm_pbb_te_mcc_send	Sends maintenance communication channel frames
cfm_pbb_te_remove_ma	Removes a maintenance association
cfm_pbb_te_remove_md	Removes a maintenance domain
cfm_pbb_te_remove_mep	Removes a maintenance end point
cfm_pbb_te_remove_rmep	Removes a maintenance association
cfm_pbb_te_send_1dm	Sends one-way delay messages
cfm_pbb_te_send_dmm	Sends two-way delay messages
cfm_pbb_te_send_lmm	Sends loss measurement management messages
cfm_pbb_te_send_mcast_lb	Sends multicast loopback messages

Function	Description
cfm_pbb_te_send_ping	Sends multicast loopback messages
cfm_pbb_te_send_ping2	Sends multicast loopback messages
cfm_pbb_te_send_traceroute	Sends a traceroute message
cfm_pbb_te_send_tst	Sends test frames
cfm_pbb_te_throughput_rx_enable	Enables throughput frame reception
cfm_pbb_te_throughput_send_frame	Sends a burst of frames of increasing sizes
cfm_pbb_te_tst_set_testing_status	Sets the testing state
cfm_pbb_te_vsm_send	Sends vendor-specific (VSM) OAM frames

Include Files

To use the functions in this chapter, you must include `onmd/cfm/cfm_pbb_te_api.h`.

cfm_pbb_te_1dm_rx_enable

This function enables one-way delay measurement reception for a MEP on a PBB-TE bridge and sets the duration of frame reception.

Syntax

```
struct cfm_1dm_rx *
cfm_pbb_te_1dm_rx_enable (struct onmd_bridge *bridge,
                          u_int8_t *md_name, u_int32_t te_sid,
                          u_int32_t mpid, u_int16_t duration)
```

Input Parameters

`bridge`	Identity of the bridge
`md_name`	Maintenance domain name
`te_sid`	TE service instance ID <1-42949675>
`mpid`	Source MEP ID <1-8191>
`duration`	Duration of message reception <5-60>

Return Values

Pointer to the `cfm_1dm_rx` structure when the function succeeds

NULL when the function fails

cfm_pbb_te_add_ma

This function adds a maintenance association (MA) to a PBB-TE bridge for a TE service ID, with the MIP creation permissions, and MA name type.

Syntax

```
s_int32_t
cfm_pbb_te_add_ma (struct cfm_md *md, u_int8_t *ma_name, u_int32_t te_sid,
                   u_int8_t mip_permission, u_int8_t ma_name_type)
```

Input Parameters

`md` Maintenance domain

`ma_name` Maintenance association name

`te_sid` TE service instance ID <1-42949675>

`mip_permission` One of the following values from the `cfm_mhf_creation` enum in `onmd/cfm/cfmd.h`:

`CFM_DEF_MHF_NONE`
No MHFs can be created for this VID

`CFM_DEF_MHF_DEFAULT`
MHFs can be created on this VID on any bridge port through which this VID can pass

`CFM_DEF_MHF_EXPLICIT`
MHFs can be created for this VID only on bridge ports through which this VID can pass, and only if a MEP is created at some lower MD level

`CFM_DEF_MHF_DEFER`
The creation of MHFs is controlled by the MD variable dot1agCfmMaCompMhfCreation

`ma_name_type` Format of the MA name; one of the following values from `onmd/cfm/cfmd.h`:

`CFM_MA_PRIMARY_VID`
Primary VLAN ID as the MA name

`CFM_MA_CHAR_STRING`
String as the MA name

`CFM_MA_TWO_OCTET_INTERGER`
Integer as MA name <0-65535>

`CFM_MA_RFC_2685_VPN_ID`
VPN ID as the MA name

Return Values

CFM_API_ERR_INTERNAL when the MD is not found

CFM_API_ERR_MA_EXISTS when the MA already exists

CFM_API_ERR_MEM when memory allocation fails

CFM_API_ERR_INVALID_MA_NAME_TYPE when `ma_name_type` is invalid

CFM_API_ERR_BRIDGE_NOT_FOUND when the bridge port list is not found

CFM_API_ERR_PBB_TE_SID_NOT_FOUND when `te_sid` is not found

CFM_API_ERR_NONE when the function succeeds

cfm_pbb_te_add_md

This function adds a maintenance domain to a PBB-TE bridge, with the MD level, MD name type, and the MIP creation permissions.

Syntax

```
s_int32_t
cfm_pbb_te_add_md (struct onmd_bridge *bridge, u_int8_t *md_name,
                   u_int16_t level, u_int8_t mip_permission,
                   u_int8_t md_name_type, **ret_md)
```

Input Parameters

`bridge` Identity of the bridge

`md_name` Maintenance domain name

`level` Maintenance domain level <0-7>

`mip_permission` One of the following values from the `cfm_mhf_creation` enum in `onmd/cfm/cfmd.h`:

`CFM_DEF_MHF_NONE`
No MHFs can be created for this VID

`CFM_DEF_MHF_DEFAULT`
MHFs can be created on this VID on any bridge port through which this VID can pass

`CFM_DEF_MHF_EXPLICIT`
MHFs can be created for this VID only on bridge ports through which this VID can pass, and only if a MEP is created at some lower MD level

`md_name_type` Format of the MD name; one of the following values from `onmd/cfm/cfmd.h`:

`CFM_NO_MD_NAME`
No MD name present

`CFM_DNS_NAME`
DNS name

`CFM_MD_CHAR_STRING`
Character string

`CFM_MAC_ADDR_TWO_OCTET_INTEGER`
MAC address + 2-octet integer

Return Values

CFM_API_ERR_INVALID_ARG when the MD name is not found

CFM_API_ERR_BRIDGE_NOT_FOUND when the bridge is not found

CFM_API_ERR_INTERNAL when the MD list is not found

CFM_API_ERR_MD_NAME_NOT_NO_NAME when the MD name type is `CFM_NO_MD_NAME`

CFM_API_ERR_INVALID_DNS_MD_NAME when the invalid DNS type of MD name

CFM_API_ERR_INVALID_MAC when the MAC type of MD name is invalid

CFM_API_ERR_INVALID_NAME_LEN_ITU_T when the length of an ITU type MD name is invalid

CFM_API_ERR_INVALID_NAME_ITU_T when the ITU type MD name is invalid

CFM_API_ERR_INVALID_MD_NAME_TYPE when the MD name type is not string

CFM_API_ERR_MEM when memory allocation fails

CFM_API_ERR_MD_EXISTS when the MD already exists

CFM_API_ERR_NONE when the function succeeds

cfm_pbb_te_add_mep

This function adds a TE-SID MEP to a PBB-TE bridge in an MD, with a MEP ID and direction.

Syntax

```
s_int32_t
cfm_pbb_te_add_mep (struct onmd_bridge *bridge, u_int16_t id, u_int8_t *md_name,
                    u_int32_t te_sid, u_int8_t *dir, struct interface *ifp)
```

Input Parameters

`bridge`	Identity of the bridge
`id`	ID of the MEP <1-8191>
`md_name`	Maintenance domain name
`te_sid`	TE service instance ID <1-42949675>
`dir`	Direction of flow, either UP or DOWN
`ifp`	Pointer to the interface

Return Values

CFM_API_ERR_BRIDGE_NOT_FOUND when the bridge is not found

CFM_API_ERR_IF_NOT_FOUND when the interface is not found

CFM_API_ERR_INTERNAL when the MD list is not found

CFM_API_ERR_MD_NOT_FOUND when the MD is not found

CFM_API_ERR_PBB_TE_DOWN_MEP_NOT_VALID when the DOWN MEP is configured in PBB-TE

CFM_API_ERR_IF_NOT_BOUND when the bridge port list is not found

CFM_API_ERR_IF_NOT_FOUND when the interface is not found

CFM_API_ERR_MAC_ADDR when the MAC address is invalid

CFM_API_ERR_AGGREGATED_PORT when the port is aggregated

CFM_API_ERR_PBB_TE_SID_NOT_FOUND when the TE-SID does not exist on the CBP

CFM_API_ERR_MA_NOT_FOUND when the MA is not found

CFM_API_ERR_MA_MEP_EXISTS when the MEP already exists

CFM_API_ERR_PORT_MEP_EXISTS when the MEP already exists

CFM_API_ERR_MEM when memory allocation fails

CFM_API_ERR_NONE when the function succeeds

cfm_pbb_te_add_rmep

This function adds a remote MEP to a PBB-TE MA in an MD for the backbone bridge.

Syntax

```
s_int32_t
cfm_pbb_te_add_rmep (struct cfm_md *md, u_int32_t rmepid, u_int32_t te_sid,
    u_int8_t *mac, u_int8_t flags)
```

Input Parameters

`md`	Maintenance domain
`rmepid`	ID of the remote MEP <1-8191>
`te_sid`	TE service instance ID <1-42949675>
`mac`	MAC address for the remote MEP
`flags`	Whether the remote MEP is a configured remote MEP

Return Values

CFM_API_ERR_INVALID_ARG when the MD is not found

CFM_API_ERR_MA_NOT_FOUND when the MA is not found

CFM_API_ERR_PBB_TE_MAC_MISMATCH when there is mismatch in the MAC address

CFM_API_ERR_PBB_TE_ESP_NOT_FOUND when the ESP is not found

CFM_API_ERR_NONE when the function succeeds

cfm_pbb_te_ais_disable

This function disables AIS on a PBB-TE bridge.

Syntax

```
s_int8_t
cfm_pbb_te_ais_disable (struct onmd_bridge *bridge, u_int32_t mepid,
                        u_int8_t *md_name, u_int32_t te_sid)
```

Input Parameters

`bridge`	Identity of the bridge
`mepid`	Source MEP ID <1-8191>
`md_name`	Maintenance domain name
`te_sid`	TE service instance ID <1-42949675>

Return Values

CFM_FAILURE when the function fails

CFM_SUCCESS when the function succeeds

cfm_pbb_te_ais_enable

This function enables Alarm Indicator Signaling (AIS) between a local and a remote MEP on a PBB-TE bridge.

Syntax

```
s_int8_t
cfm_pbb_te_ais_enable (struct onmd_bridge *bridge, u_int32_t mepid,
                       u_int8_t ais_type, u_int8_t *md_name, u_int8_t *rmac,
                       s_int8_t destLevel, u_int32_t te_sid)
```

Input Parameters

`bridge` Identity of the bridge

`mepid` Source MEP ID <1-8191>

`ais_type` AIS signal type; one of these values from `onmd/cfm/cfmd.h`:

`CFM_UCAST_FRAME`
Unicast

`CFM_MCAST_FRAME`
Multicast

`md_name` Maintenance domain name

`rmac` MAC address of the remote MEP

`destLevel` Level of the destination TE service instance <1-7>

`te_sid` TE service instance ID <1-42949675>

Return Values

CFM_FAILURE when the function fails

CFM_SUCCESS when the function succeeds

cfm_pbb_te_ais_set_tx_interval

This function sets the transmission interval for AIS between a MEP and a TE service instance on a PBB-TE bridge.

Syntax

```
s_int8_t
cfm_pbb_te_ais_set_tx_interval (struct onmd_bridge *bridge,
                                u_int8_t tx_interval, u_int32_t mepid,
                                u_int8_t *md_name, u_int32_t te_sid)
```

Input Parameters

`bridge` Identity of the bridge

`tx_interval` Transmission interval desired between frames

`mepid` Source MEP ID <1-8191>

`md_name` Maintenance domain name

`te_sid` TE service instance ID <1-42949675>

Return Values

CFM_FAILURE when the function fails

CFM_SUCCESS when the function succeeds

cfm_pbb_te_cc_mcast_disable

This function disables multicast continuity check messages on a traffic-engineered provider backbone bridge.

Syntax

```
s_int32_t
cfm_pbb_te_cc_mcast_disable (struct onmd_bridge *bridge,
                             u_int8_t *md_name,
                             u_int32_t te_sid, struct interface *ifp)
```

Input Parameters

`bridge`	Identity of the bridge
`md_name`	Maintenance domain name
`te_sid`	TE service instance ID <1-42949675>
`ifp`	Pointer to interface

Return Values

CFM_BRIDGE_NOT_FOUND when the bridge is not found

CFM_API_ERR_IF_NOT_BOUND when the bridge port is not found

CFM_MD_LIST_NOT_FOUND when the MD list is not found

CFM_DOMAIN_NOT_FOUND when the MD is not found

CFM_MA_NOT_FOUND when the MA is not found

CFM_API_ERR_PBB_TE_SID_NOT_FOUND when the `te_sid` is not found

CFM_API_ERR_PBB_TE_SID_NOT_PT_TO_MPT when the `te_sid` is not point-to-multipoint

CFM_API_ERR_MEP_NOT_FOUND when the MEP is not found

CFM_API_ERR_CC_NOT_ENABLED when the MA is not CC enabled

CFM_API_ERR_NONE when the function succeeds

cfm_pbb_te_cc_mcast_enable

This function enables multicast continuity check messages (CCM) in a point-to-multipoint network on a traffic-engineered provider backbone bridge.

Syntax

```
s_int32_t
cfm_pbb_te_cc_mcast_enable (struct onmd_bridge *bridge,
                            u_int8_t *md_name,
                            u_int32_t te_sid, struct interface *ifp)
```


Input Parameters

`bridge`	Identity of the bridge
`md_name`	Maintenance domain name
`te_sid`	TE service instance ID <1-42949675>
`ifp`	Pointer to interface

Return Values

CFM_BRIDGE_NOT_FOUND when the bridge is not found

CFM_DOMAIN_NOT_FOUND when the MD is not found

CFM_API_ERR_IF_NOT_BOUND when the bridge port list is not found

CFM_MD_LIST_NOT_FOUND when the MD list is not found

CFM_API_ERR_MA_NOT_FOUND when the MA is not found

CFM_API_ERR_PBB_TE_SID_NOT_FOUND when the `te_sid` is not found

CFM_API_ERR_PBB_TE_SID_NOT_PT_TO_MPT when the `te_sid` is not point-to-multipoint

CFM_API_ERR_MEP_NOT_FOUND when the MEP is not found

CFM_API_ERR_MEP_CCM_ALREADY_ENABLED when the CCM is already enabled

CFM_API_ERR_PBB_TE_ESP_NOT_FOUND when the ESP-DA is not same as the destination MAC

CFM_API_ERR_NONE when the function succeeds

cfm_pbb_te_cc_unicast_disable

This function disables unicast continuity check messages on a traffic-engineered provider backbone bridge.

Syntax

```
s_int32_t
cfm_pbb_te_cc_unicast_disable (struct onmd_bridge *bridge, u_int8_t *md_name,
                               u_int32_t te_sid, u_int32_t mpid, u_int32_t rmpid,
                               struct interface *ifp)
```

Input Parameters

`bridge`	Identity of the bridge
`md_name`	Maintenance domain name
`te_sid`	TE service instance ID <1-42949675>
`mpid`	ID of the source MEP <1-8191>
`rmpid`	ID of the remote MEP <1-8191>
`ifp`	Pointer to interface

Return Values

CFM_BRIDGE_NOT_FOUND when the bridge is not found

CFM_MD_LIST_NOT_FOUND when the MD list is not found

CFM_API_ERR_IF_NOT_BOUND when the bridge port list is not found

CFM_DOMAIN_NOT_FOUND when the MD is not found

CFM_MA_NOT_FOUND when the MA is not found

CFM_API_ERR_PBB_TE_SID_NOT_FOUND when the `te_sid` is not found

CFM_MEP_NOT_FOUND when the MEP is not found

CFM_MCAST_ENABLED when already multicast enabled

CFM_RMEP_NOT_FOUND when the RMEP is not found

CFM_API_ERR_NONE when the function succeeds

cfm_pbb_te_cc_unicast_enable

This function enables unicast continuity check messages on a traffic-engineered provider backbone bridge.

Syntax

```
s_int32_t
cfm_pbb_te_cc_unicast_enable (struct onmd_bridge *bridge,
                              u_int8_t *md_name, u_int32_t te_sid,
                              u_int32_t mpid, u_int32_t rmpid,
                              struct interface *ifp)
```

Input Parameters

`bridge`	Identity of the bridge
`md_name`	Maintenance domain name
`te_sid`	TE service instance ID <1-42949675>
`mpid`	ID of the source MEP <1-8191>
`rmpid`	ID of the remote MEP <1-8191>
`ifp`	Pointer to interface

Return Values

CFM_BRIDGE_NOT_FOUND when the bridge is not found

CFM_API_ERR_IF_NOT_BOUND when the bridge port is not found

CFM_MD_LIST_NOT_FOUND when the MD list is not found

CFM_DOMAIN_NOT_FOUND when the MD is not found

CFM_MA_NOT_FOUND when the MA is not found

CFM_API_ERR_PBB_TE_SID_NOT_FOUND when the TE-SID is not found

CFM_MEP_NOT_FOUND when the MEP is not found

CFM_API_ERR_PBB_TE_SID_NOT_PT_TO_PT when the `te_sid` is not point-to-point

CFM_API_ERR_MEP_CCM_ALREADY_ENABLED when the CCM is already unicast/multicast enabled

CFM_RMEP_NOT_FOUND when the RMEP is not found

CFM_RMEP_MAC_NOT_FOUND when the RMEP MAC is not found

CFM_API_ERR_PBB_TE_ESP_NOT_FOUND when the packet is sent with an invalid MAC

CFM_API_ERR_NONE when the function succeeds

cfm_pbb_te_exm_send

This function sends experimental (EXM) OAM frames between a local and a remote MEP on a PBB-TE bridge.

Syntax

```
s_int32_t
cfm_pbb_te_exm_send (struct onmd_bridge *bridge, u_int8_t *md_name,
                     u_int32_t te_sid, u_int32_t mepid, u_int32_t rmepid)
```

Input Parameters

`bridge`	Identity of the bridge
`md_name`	Maintenance domain name
`te_sid`	TE service instance ID <1-42949675>
`mepid`	Source MEP ID <1-8191>
`rmepid`	Remote MEP ID <1-8191>

Return Values

CFM_BRIDGE_NOT_FOUND when the bridge is not found

CFM_MD_LIST_NOT_FOUND when the MD list is not found

CFM_DOMAIN_NOT_FOUND when the MD is not found

CFM_MA_NOT_FOUND when the MA is not found

CFM_MEP_NOT_FOUND when the MEP is not found

CFM_API_ERR_PBB_TE_SID_NOT_PT_TO_PT when the TE-SID is not point-to-point

CFM_MEMORY_NOT_ALLOCATED when memory allocation fails

CFM_RMEP_NOT_FOUND when the RMEP is not found

CFM_API_ERR_PBB_TE_ESP_NOT_FOUND when the packet is sent with an invalid MAC

CFM_API_ERR_NONE when the function succeeds

cfm_pbb_te_mcc_send

This function sends Maintenance Communication Channel (MCC) frames between a local and a remote MEP on a PBB-TE bridge.

Syntax

```
s_int32_t
cfm_pbb_te_mcc_send (struct onmd_bridge *bridge,
                     u_int8_t *md_name, u_int32_t te_sid, u_int32_t mepid,
                     u_int32_t rmepid,u_int8_t frame_type)
```

Input Parameters

`bridge`	Identity of the bridge
`md_name`	Maintenance domain name
`te_sid`	TE service instance ID <1-42949675>
`mepid`	Source MEP ID <1-8191>

`rmepid` Remote MEP ID <1-8191>

`frame_type` Type of MCC frames to transmit; one of these values from `onmd/cfm/cfmd.h`:

`CFM_UCAST_FRAME`
Unicast

`CFM_MCAST_FRAME`
Multicast

Return Values

CFM_BRIDGE_NOT_FOUND when the bridge is not found

CFM_MD_LIST_NOT_FOUND when the MD list is not found

CFM_DOMAIN_NOT_FOUND when the MD is not found

CFM_MA_NOT_FOUND when the MA is not found

CFM_MEP_NOT_FOUND when the MEP is not found

CFM_API_ERR_PBB_TE_SID_NOT_PT_TO_PT when the TE-SID is not point-to-point

CFM_MEMORY_NOT_ALLOCATED when memory allocation fails

CFM_RMEP_NOT_FOUND when the RMEP is not found

CFM_API_ERR_PBB_TE_ESP_NOT_FOUND when the packet is sent with an invalid MAC

CFM_API_ERR_NONE when the function succeeds

cfm_pbb_te_remove_ma

This function removes an MA from a PBB-TE bridge by identifying the MD, MA name, TE service ID, and MIP creation permissions.

Syntax

```
s_int32_t
cfm_pbb_te_remove_ma (struct cfm_md *md, u_int8_t *ma_name,
    u_int32_t te_sid, u_int8_t mip_permission)
```

Input Parameters

`md` Maintenance domain

`ma_name` Maintenance association name

`te_sid` TE service instance ID <1-42949675>

`mip_permission` One of the following values from the `cfm_mhf_creation` enum in `onmd/cfm/cfmd.h`:

`CFM_DEF_MHF_NONE`
No MHFs can be created for this VID

`CFM_DEF_MHF_DEFAULT`
MHFs can be created on this VID on any bridge port through which this VID can pass

`CFM_DEF_MHF_EXPLICIT`
MHFs can be created for this VID only on bridge ports through which this VID can pass, and only if a MEP is created at some lower MD level

`CFM_DEF_MHF_DEFER`

The creation of MHFs is controlled by the MD variable dot1agCfmMaCompMhfCreation

Return Values

CFM_API_ERR_INTERNAL when the MD is not found

CFM_API_ERR_MA_NOT_FOUND when the MA is not found

CFM_API_ERR_NONE when the function succeeds

cfm_pbb_te_remove_md

This function removes a maintenance domain from a PBB-TE bridge and MD level.

Syntax

```
s_int32_t
cfm_pbb_te_remove_md (struct onmd_bridge *bridge, u_int8_t *md_name,
                      u_int16_t level, u_int8_t mip_permission)
```

Input Parameters

`bridge` Identity of the bridge

`md_name` Maintenance domain name

`level` Maintenance domain level <0-7>

`mip_permission` One of the following values from the `cfm_mhf_creation` enum in `onmd/cfm/cfmd.h`:

`CFM_DEF_MHF_NONE`
No MHFs can be created for this VID

`CFM_DEF_MHF_DEFAULT`
MHFs can be created on this VID on any bridge port through which this VID can pass

`CFM_DEF_MHF_EXPLICIT`
MHFs can be created for this VID only on bridge ports through which this VID can pass, and only if a MEP is created at some lower MD level

`CFM_DEF_MHF_DEFER`
The creation of MHFs is controlled by the MD variable dot1agCfmMaCompMhfCreation

Return Values

CFM_API_ERR_MD_NOT_FOUND when the MD is not found

CFM_API_ERR_BRIDGE_NOT_FOUND when the bridge is not found

CFM_API_ERR_INTERNAL when the MD list is not found

CFM_API_ERR_NONE when the function succeeds

cfm_pbb_te_remove_mep

This function removes a MEP from a PBB-TE bridge and identify the direction of flow (UP or DOWN).

Syntax

```
s_int32_t
cfm_pbb_te_remove_mep (struct onmd_bridge *bridge, u_int16_t id,
```

```
u_int8_t *md_name, u_int32_t te_sid,
u_char *dir, struct interface *ifp)
```

Input Parameters

`bridge`	Identity of the bridge
`md_name`	Maintenance domain name
`id`	MEP ID <1-8191>
`te_sid`	TE service instance ID <1-42949675>
`dir`	Direction of flow for the MEP, Up or Down
`ifp`	Pointer to the interface

Return Values

CFM_API_ERR_BRIDGE_NOT_FOUND when the bridge is not found

CFM_API_ERR_IF_NOT_FOUND when the interface is not found

CFM_API_ERR_INTERNAL when the MD list is not found

CFM_API_ERR_MD_NOT_FOUND when the MD is not found

CFM_API_ERR_MA_NOT_FOUND when the MA is not found

CFM_API_ERR_MEP_NOT_FOUND when the MEP is not found

RESULT_OK when the function succeeds

cfm_pbb_te_remove_rmep

This function removes a remote MEP from a PBB-TE bridge by identifying the MD, remote MEP, TE service ID and MAC address.

Syntax

```
s_int32_t
cfm_pbb_te_remove_rmep (struct cfm_md *md, u_int32_t rmepid, u_int32_t te_sid,
    u_int8_t *mac)
```

Input Parameters

`md`	Maintenance domain
`rmepid`	ID of the remote MEP <1-8191>
`te_sid`	TE service instance ID <1-42949675>
`mac`	MAC address of the remote MEP

Return Values

CFM_API_ERR_INVALID_ARG when the MD is not found

CFM_API_ERR_MA_NOT_FOUND when the MA is not found

CFM_API_ERR_RMEP_NOT_FOUND when the remote MEP cannot be found

CFM_API_ERR_NONE when the function succeeds

cfm_pbb_te_send_1dm

This function sends one-way delay messages between a source and a remote MEP on a PBB-TE bridge, and specify the duration, transmission interval and frame type.

Note: You can only call this function if ZebOS-XP is built with ITU Y.1731 support.

Syntax

```
struct cfm_1dm_tx *
cfm_pbb_te_send_1dm (struct onmd_bridge *bridge, u_int8_t *md_name,
                     u_int32_t te_sid, u_int32_t mpid,
                     u_int32_t rmpid, u_int16_t duration,
                     u_int8_t tx_interval, u_int8_t frame_type)
```

Input Parameters

`bridge`	Identity of the bridge
`md_name`	Maintenance domain name
`te_sid`	TE service instance ID <1-42949675>
`mpid`	Source MEP ID <1-8191>
`rmpid`	Remote MEP ID <1-8191>
`duration`	Duration of message transmission <5-60>
`tx_interval`	Transmission interval desired between frames <1-4>
`frame_type`	Type of MCC frames to transmit; one of these values from `onmd/cfm/cfmd.h`:
`CFM_UCAST_FRAME`	Unicast
`CFM_MCAST_FRAME`	Multicast

Return Values

Pointer to the `cfm_1dm_tx` structure when the function succeeds

NULL when the function fails

cfm_pbb_te_send_dmm

This function sends two-way delay messages between a source and a remote MEP on a PBB-TE bridge.

Note: You can only call this function if ZebOS-XP is built with ITU Y.1731 support.

Syntax

```
struct cfm_dmm *
cfm_pbb_te_send_dmm (struct onmd_bridge *bridge, u_int8_t *md_name,
                     u_int32_t te_sid, u_int32_t mepid, u_int32_t rmepid,
                     u_int16_t duration, u_int8_t tx_interval,
                     u_int8_t frame_type)
```

Input Parameters

`bridge`	Identity of the bridge

`md_name`	Maintenance domain name
`te_sid`	TE service instance ID <1-42949675>
`mepid`	Source MEP ID <1-8191>
`rmepid`	Remote MEP ID <1-8191>
`duration`	Duration of message transmission <5-60>
`tx_interval`	Transmission interval desired between frames <1-4>
`frame_type`	Type of MCC frames to transmit; one of these values from `onmd/cfm/cfmd.h`:
`CFM_UCAST_FRAME`	Unicast
`CFM_MCAST_FRAME`	Multicast

Return Values

Pointer to the `cfm_dmm` structure when the function succeeds

NULL when the function fails

cfm_pbb_te_send_lmm

This function sends loss measurement management messages between local and remote MEPs on a PBB-TE bridge.

Syntax

```
struct cfm_lm *
cfm_pbb_te_send_lmm (struct onmd_bridge *bridge, u_int32_t mepid,
                     u_int32_t rmepid, u_int8_t lm_type,
                     u_int16_t duration, u_int8_t tx_interval,
                     u_int8_t *md_name, u_int32_t te_sid)
```

Input Parameters

`bridge`	Identity of the bridge
`mepid`	Source MEP ID <1-8191>
`rmepid`	Remote MEP ID <1-8191>
`lm_type`	Type of loss measurement
`duration`	Duration of message transmission <5-60>
`tx_interval`	Transmission interval desired between frames <1-4>
`md_name`	Maintenance domain name
`te_sid`	TE service instance ID <1-42949675>

Return Values

Pointer to the `cfm_lm` structure when the function succeeds

NULL when the function fails

cfm_pbb_te_send_mcast_lb

This function sends multicast loopback messages from a MEP on a PBB-TE bridge to a TE service instance, designate TLV type, TLV indictor, and specify whether messages should be sent recursively.

Syntax

```
struct cfm_lb *
cfm_pbb_te_send_mcast_lb (struct onmd_bridge *bridge, u_int8_t *md_name,
                          u_int32_t mepid, u_int8_t tlv_type,
                          u_int16_t tlv_indicator, u_int32_t te_sid,
                          u_int8_t recursive_mcast)
```

Input Parameters

`bridge`	Identity of the bridge
`md_name`	Maintenance domain name
`mepid`	Source MEP ID <1-8191>
`tlv_type`	Type of TLV data requested
`tlv_indicator`	Test pattern: one of these values from the `test_pattern_indicator` enum in `onmd/cfm/cfm_lb.h`:
	`TEST_TLV_PATTERN_1`
	`TEST_TLV_PATTERN_2`
	`TEST_TLV_PATTERN_3`
	`TEST_TLV_PATTERN_4`
`te_sid`	TE service instance ID <1-42949675>
`recursive`	Send multicast messages recursively

Return Values

Pointer to the `cfm_lb` structure when the function succeeds

NULL when the function fails

cfm_pbb_te_send_ping

This function sends a loopback message between a source and a remote MEP on a PBB-TE bridge by identifying the MD name and TE service instance ID.

This function has two signatures that are identical except the second has additional parameters for TLV data. Use the second signature if you have a ZebOS-XP build with ITU Y.1731 support, otherwise use the first signature.

Syntax

```
struct cfm_lb *
cfm_pbb_te_send_ping (struct onmd_bridge *bridge, u_int32_t source_mepid,
                      u_int32_t rmepid, u_int8_t *md_name, u_int32_t te_sid)
struct cfm_lb *
cfm_pbb_te_send_ping (struct onmd_bridge *bridge, u_int32_t source_mepid,
                      u_int32_t rmepid, u_int8_t *md_name, u_int8_t tlv_type,
                      u_int16_t tlv_indicator, u_int32_t te_sid)
```

Input Parameters

`bridge`	Identity of the bridge
`source_mepid`	Source MEP ID <1-8191>
`rmepid`	Remote MEP ID <1-8191>
`md_name`	Maintenance domain name
`tlv_type`	Type of TLV data
`tlv_indicator`	Test pattern: one of these values from the `test_pattern_indicator` enum in `onmd/cfm/cfm_lb.h`:
	`TEST_TLV_PATTERN_1`
	`TEST_TLV_PATTERN_2`
	`TEST_TLV_PATTERN_3`
	`TEST_TLV_PATTERN_4`
`te_sid`	TE service instance ID <1-42949675>

Return Values

Pointer to the `cfm_lb` structure when the function succeeds

NULL when the function fails

cfm_pbb_te_send_ping2

This function sends a loopback message between a source and a remote MEP on a PBB-TE bridge by identifying the MD name and TE service instance ID.

This function has two signatures that are identical except the second has additional parameters for TLV data. Use the second signature if you have a ZebOS-XP build with ITU Y.1731 support, otherwise use the first signature.

Syntax

```
struct cfm_lb *
cfm_pbb_te_send_ping2(struct onmd_bridge *bridge, u_int8_t *md_name,
                      u_int32_t te_sid, u_int16_t reverse_vid, u_int8_t *mac,
                      u_int16_t src_mep_id)
struct cfm_lb *
cfm_pbb_te_send_ping2(struct onmd_bridge *bridge, u_int8_t *md_name,
                      u_int32_t te_sid, u_int16_t reverse_vid, u_int8_t *mac,
                      u_int16_t src_mep_id, u_int8_t tlv_type, u_int16_t tlv_indicator)
```

Input Parameters

`bridge`	Identity of the bridge
`md_name`	Maintenance domain name
`te_sid`	TE service instance ID <1-42949675>
`reverse_vid`	ID of the VLAN to send in the response
`mac`	MAC address of the remote MEP
`src_mep_id`	ID of the source MEP
`tlv_type`	Type of TLV data requested

`tlv_indicator`	Test pattern: one of these values from the `test_pattern_indicator` enum in `onmd/cfm/cfm_lb.h`:
	`TEST_TLV_PATTERN_1`
	`TEST_TLV_PATTERN_2`
	`TEST_TLV_PATTERN_3`
	`TEST_TLV_PATTERN_4`

Return Values

Pointer to the `cfm_lb` structure when the function succeeds

NULL when the function fails

cfm_pbb_te_send_traceroute

This function sends a traceroute message from a PBB-TE MEP with the TE-SID and the reverse VLAN which is used by remote MEP or MIP to be returned.

Syntax

```
struct cfm_lt *
cfm_pbb_te_send_traceroute (struct onmd_bridge *bridge, u_int8_t *mac,
                            u_int8_t *md_name, u_int32_t te_sid, u_int16_t reverse_vid)
```

Input Parameters

`bridge`	Identity of the bridge
`mac`	MAC address
`md_name`	Maintenance domain name
`bridge`	Identity of the bridge
`te_sid`	TE service instance ID <1-42949675>
`reverse_vid`	ID of the VLAN to send in the response

Return Values

Pointer to the `cfm_lt` structure when the function succeeds

NULL when the function fails

cfm_pbb_te_send_tst

This function sends test (TST) frames between a source and a remote MEP on a PBB-TE bridge, and configure frame characteristics.

Syntax

```
struct cfm_tst *
cfm_pbb_te_send_tst (struct onmd_bridge *bridge, u_int32_t mepid,
                     u_int32_t rmepid, u_int8_t *md_name, u_int8_t frame_type,
                     u_int8_t pat_indicator, u_int8_t duration,
                     s_int8_t tx_interval, u_int32_t te_sid,
                     u_int8_t recursive, u_int8_t lck_tx_interval,
                     u_int32_t lck_rmepid, u_int8_t lck_frame_type)
```

Input Parameters

`bridge`	Identity of the bridge
`mepid`	Source MEP ID <1-8191>
`rmepid`	Remote MEP ID <1-8191>
`md_name`	Maintenance domain name
`frame_type`	Type of TST frames to transmit
`pat_indicator`	Test pattern indicator
`duration`	Duration of message transmission <5-60>
`tx_interval`	Transmission interval between frames <1-10>
`te_sid`	TE service instance ID <1-42949675>
`recursive`	Send TST frames recursively
`lck_tx_interval`	Lock (LCK) frame transmission interval <1-10>
`lck_rmepid`	ID of remote MEP for LCK frames <1-8191>
`lck_frame_type`	Type of LCK frame; one of these values from `onmd/cfm/cfmd.h`:
`CFM_UCAST_FRAME`	Unicast
`CFM_MCAST_FRAME`	Multicast

Return Values

Pointer to the `cfm_tst` structure when the function succeeds

NULL when the function fails

cfm_pbb_te_throughput_rx_enable

This function enables throughput frame reception on a local MEP on a PBB-TE bridge.

Syntax

```
struct cfm_tput_reception *
cfm_pbb_te_throughput_rx_enable (struct onmd_bridge *bridge,
                                 u_int32_t mepid, u_int8_t duration,
                                 u_int8_t *md_name, u_int32_t te_sid)
```

Input Parameters

`bridge`	Identity of the bridge
`mepid`	Source MEP ID <1-8191>
`duration`	Duration of throughput-frame reception <1-10>
`md_name`	Maintenance domain name
`te_sid`	TE service instance ID <1-42949675>

Return Values

Pointer to the `cfm_tput_reception` structure when the function succeeds

NULL when the function fails

cfm_pbb_te_throughput_send_frame

This function sends a burst of frames of increasing sizes between a local and a remote MEP on a PBB-TE bridge.

Syntax

```
s_int8_t
cfm_pbb_te_throughput_send_frame (struct onmd_bridge *bridge,
                                  u_int32_t mepid, u_int32_t rmepid,
                                  u_int8_t *md_name, u_int32_t te_sid)
```

Input Parameters

`bridge`	Identity of the bridge
`mepid`	Source MEP ID <1-8191>
`rmepid`	Remote MEP ID <1-8191>
`md_name`	Maintenance domain name
`te_sid`	TE service instance ID <1-42949675>

Return Values

CFM_FAILURE when the function fails

CFM_SUCCESS when the function succeeds

cfm_pbb_te_tst_set_testing_status

This function sets the testing state for a MEP on a PBB-TE bridge.

Syntax

```
s_int32_t
cfm_pbb_te_tst_set_testing_status (struct onmd_bridge *bridge, u_int32_t mepid,
                                   u_int8_t testing_state, u_int8_t destLevel,
                                   u_int8_t *md_name, u_int32_t te_sid)
```

Input Parameters

`bridge`	Identity of the bridge
`mepid`	Source MEP ID <1-8191>
`testing_state`	Testing state; one of these values from `onmd/cfm/cfm_tst.h`:
	`CFM_MEP_IN_SERVICE_TESTING`
	Set the MEP for in-service testing
	`CFM_MEP_OUT_OF_SERVICE_TESTING`
	Set the MEP for out-of-service testing
`destLevel`	Destination maintenance level
`md_name`	Maintenance domain name
`te_sid`	TE service instance ID <1-42949675>

Return Values

CFM_FAILURE when the function fails

CFM_SUCCESS when the function succeeds

cfm_pbb_te_vsm_send

This function sends vendor-specific (VSM) OAM frames between a local and a remote MEP on a PBB-TE bridge.

Syntax

```
s_int32_t
cfm_pbb_te_vsm_send (struct onmd_bridge *bridge, u_int8_t *md_name,
                     u_int32_t te_sid, u_int32_t mepid, u_int32_t rmepid)
```

Input Parameters

`bridge`	Identity of the bridge
`md_name`	Maintenance domain name
`te_sid`	TE service instance ID <1-42949675>
`mepid`	Source MEP ID <1-8191>
`rmepid`	Remote MEP ID <1-8191>

Return Values

CFM_BRIDGE_NOT_FOUND when the bridge is not found

CFM_MD_LIST_NOT_FOUND when the MD list is not found

CFM_DOMAIN_NOT_FOUND when the MD is not found

CFM_MA_NOT_FOUND when the MA is not found

CFM_MEP_NOT_FOUND when the MEP is not found

CFM_API_ERR_PBB_TE_SID_NOT_PT_TO_PT when the `te_sid` is not point-to-point

CFM_MEMORY_NOT_ALLOCATED when memory allocation fails

CFM_RMEP_NOT_FOUND when the RMEP is not found

CFM_API_ERR_PBB_TE_ESP_NOT_FOUND when the packet is sent with an invalid MAC

CFM_API_ERR_NONE when the function succeeds

CHAPTER 6 Proactive Ethernet OAM

Proactive OAM uses dual-ended ETH-LM (Ethernet Loss Management) for performance monitoring and fault management. Proactive OAM periodically sends frames with ETH-LM information and the system computes the frame loss. Each MEP sends periodic dual-ended frames with ETH-LM information to its peer MEP in a point-to-point ME to facilitate frame loss measurements at the peer MEP. Each MEP terminates the dual-ended frames with ETH-LM information and makes the near-end and far-end loss measurements. Once enabled, the CCM frame continues to carry ETH-LM information until ETH-LM is disabled for the MEP. When configured for proactive loss measurement, a MEP periodically transmits CCM frames with the following information elements:

- TxFCf: This is the value of the local counter TxFCl at the time of transmission of the CCM frame.
- RxFCb: This is the value of the local counter RxFCl at the time of reception of the last CCM frame from the peer MEP.
- TxFCb: This is the value of TxFCf in the last received CCM frame from the peer MEP.

A CCM PDU transmits a period value equal to the CCM transmission period, which is configured in the performance monitoring application at the transmitting MEP. The receiving MEP detects an unexpected period defect condition if the CCM transmission period is not the same as the configured value.

System Architecture

The hardware frame count is simulated using the frame count simulator CLI, which is a thread that runs in the background and periodically increments the counter.

CCM PDU Format

The information elements carried in CCM to support dual-ended ETH-LM include:

- TxFCf: This element is a 4-octet field that carries the value of the counter of in-profile data frames transmitted by the MEP towards its peer MEP at the time of CCM frame transmission.
- RxFCb: This element is a 4-octet field that carries the value of the counter of in-profile data frames received by the MEP from its peer MEP at the time of receiving the last CCM frame from that peer MEP.
- TxFCb: This element is a 4-octet field that carries the value of the TxFCf field in the last CCM frame received by the MEP from its peer MEP.

System Design

When CCM is disabled for a MEP, the enabled ETH-LM is also disabled. The next time the user enables CCM, they have to also enable ETH-LM. This is because the user probably also enabled multicast CCM when they enabled CCM on the MEP, so ETH-LM calculations is not done unless ETH-LM is also enabled. Users can enable ETH-LM only for unicast CCMs. Once the user disables CCM, ETH-LM is also internally disabled. This is done because the user might enable unicast or multicast CCM the next time and ETH-LM might attempt CCM, which is a violation. Therefore, when a user disables CCM, they also disable ETH-LM. The user has to enable ETH-LM. This is because the user enabled unicast CCM when they enabled CCM on the MEP, so the ETH-LM calculations is not done unless ETH-LM is enabled explicitly. For each ETH-LM calculation that is occurs when receiving a CCM, the near-end and far-end loss measurements are logged.

Transmission of ETH-LM in CCM

Transmission of ETH-LM in CCM is accomplished within the MEP CCM transmission framework. Users can enable CCM transmission between a pair of MEPs. Once enabled, CCM sent by the MEP carries frame counters as needed

for dual-ended ETH-LM. A CCM PDU transmits a Period value equal to the CCM transmission period configured in the performance monitoring application at the transmitting MEP. The receiving MEP detects an unexpected period defect condition if the CCM transmission period is not the same as the configured value.

Reception of ETH-LM in CCM

Reception of ETH-LM in CCM is accomplished using the MEP CCM reception framework. When the ETH-LM information is received in an Eth-CCM frame, the information is processed only if the MEP is enable for dual-ended ETH-LM. Otherwise, the ETH-LM is not used. When configured for proactive loss measurement, a MEP uses the following values to make near-end and far-end loss measurements when it receives a CCM frame:

- Received CCM frames TxFCf, RxFCb, and TxFCb values and local counter RxFCl value when receiving this CCM frame. These values are represented as TxFCf[tc], RxFCb[tc], TxFCb[tc], and RxFCl[tc], where “tc” is the reception time of the current frame.
- Previous CCM frames TxFCf, RxFCb, and TxFCb values and local counter RxFCl value at the time the previous CCM frame was received. These values are represented as TxFCf[tp], RxFCb[tp], TxFCb[tp], and RxFCl[tp], where “tp” is the reception time of the previous frame.

  ```
  Frame Lossfar-end = |TxFCb[tc] - TxFCb[tp]| - |RxFCb[tc] - RxFCb[tp]|
  ```

  ```
  Frame Lossnear-end = |TxFCf[tc] - TxFCf[tp]| - |RxFCl[tc] - RxFCl[tp]|
  ```

- If the Period field value in the received CCM frame is different from the one configured for the MEP CCM transmission period, the MEP detects an unexpected period defect condition, in which case frame loss measurements not carried out.

CHAPTER 7 Service OAM

The ZebOS-XP Service OAM (SOAM) provides a robust set of management tools to help maintain Ethernet service networks.

Overview

Service OAM consists of Fault Management and Performance Management capabilities that easily incorporate into network elements to help support Ethernet services. OAM (Operations, Administration and Maintenance) services help manage network infrastructures and services provided across a network infrastructure. The ZebOS-XP Service OAM module provides the framework necessary for systems to operate within a MEF-compliant Metro Ethernet Networks (MENs).

SOAM requirements represent expectations of Service Providers when managing Ethernet Services within a MEN and Subscribers when managing Ethernet services across a MEN. Service OAM framework describes the high-level constructs used to model different MEN and Service components that are relevant for OAM. The framework also describes the relationship between Service OAM and the architectural constructs of Ethernet Services (ETH), Transport Service (TRAN) and Application Service (APP) Layers as identified in the MEF 4 specifications. SOAM supports OAM across a specific Class of Service (CoS) instance when multiple CoS instances are supported within an Ethernet service that need to be managed for the purposes of performance monitoring.

Features

The following are some of the features of SOAM:

- Full implementation of MEF 17 (Service OAM)
- Fault management used to detect, verify, localize, and notify faults
- Performance monitoring, including performance parameter measurements
- Auto-discovery, including discovering service aware network elements within provider networks
- Supports both Intra- and inter-provider Service OAM

System Design

By design, subscribers connect to Metro Ethernet Network (MEN) across a User Network Interface (UNI). Network elements (such as, IP, ATM, and Frame Relay elements) inside MEN interconnect through Internal Network-to-Network Interfaces (I-NNIs), and two autonomous MENs interconnect at an External NNI (E-NNI). MEN could also interconnect with other transport networks via Network Interworking NNI (NI-NNI) or with other service networks through Service Interworking NNI (SI-NNI).

Figure 7-1 displays the MEN external interfaces and associated reference points.

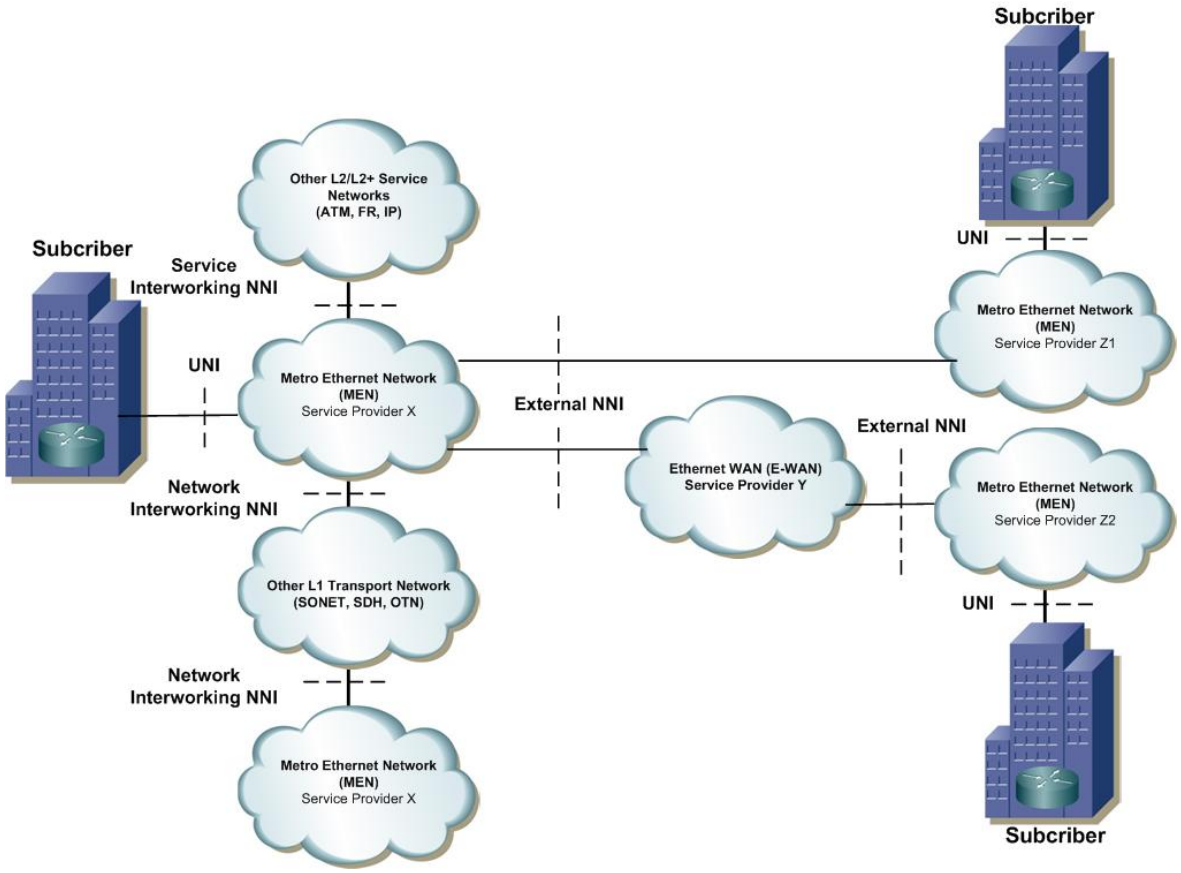


Figure 7-1: MEN External Interfaces and Reference Points

The MEN layered network model includes the data, control and management planes. These planes are present for all three Layers of this model and are called the Transport Service Layer (TRAN Layer), Ethernet Service Layer (ETH Layer) and Application Service Layer (APP Layer). Figure 7-2 displays a typical arrangement implemented by Service Providers to manage their networks and services offered across these networks. It is an overview of the requirements and framework utilized by Service OAM across all network elements (NEs). MEF 7 supports network and service management using the NMS-EMS management interface and MEF 15 provide the management requirements for NEs.

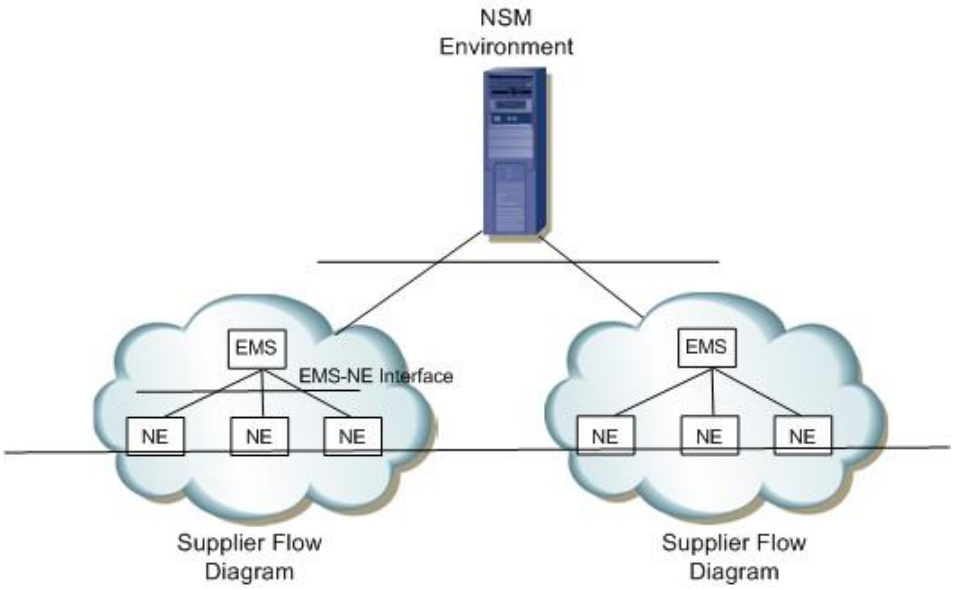


Figure 7-2: Typical Arrangement Implemented by Service Providers

Ethernet Services Layer

In the following figure, Subscriber Sites A, B, and C represent a multipoint-to-multipoint Ethernet service. Similarly, Subscriber Sites A, B, and C could represent a point-to-point service using either a single or multiple Service Provider MEN. Only components related to the Ethernet service-aware manager are relevant to the Ethernet layer OAM. An EVC is a logical construct in the Ethernet Layer, since it enables end-to-end Subscriber service instances across one or more MEN Service Providers.

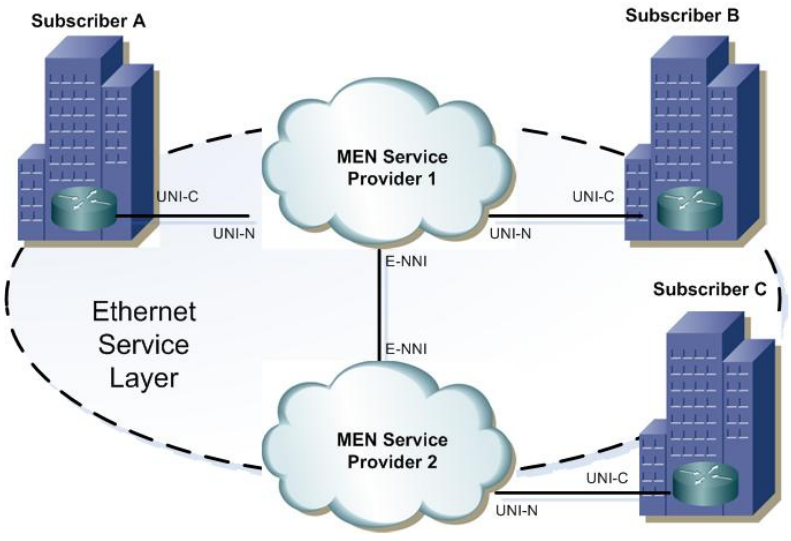


Figure 7-3: Ethernet Layer Interfaces and Reference Points

OAM Domain

The OAM domain operates within the Ethernet layer level as either a network or sub-network. It belongs to the same entity that exchanges OAM frames. Typically, each service provider and/or network operator associates with this entity's boundary. Services might occur across a single or multiple networks. The OAM domain determines the span of the OAM flow across the administrative boundary. A subscriber OAM domain may completely overlap OAM domains of multiple service providers only if the OAM domain of the service provider remains transparent to the OAM domain of the subscriber. Likewise, a subscriber OAM domain may completely overlap OAM domains of multiple network operators only if the OAM domain of the network operator remains transparent to OAM domain of the service provider.

OAM Components

The following section describes the OAM components.

Maintenance Entity (ME)

The Maintenance Entity (ME) of OAM determines the application the OAM flow. It represents an OAM entity that requires management. That is, each ME corresponds directly with the Ethernet layer and is essentially an association between two maintenance endpoints within an OAM Domain, where each endpoint corresponds to a provisioned reference point that requires management. Typically, the subscriber OAM domain includes of an ME called "Subscriber ME, and the service provider OAM domains consists of the ME marked "EVC ME". If the UNI between the subscriber and the service provider requires network management, an ME called "UNI ME" appears.

If the service provider uses facilities of two different network operators, each OAM Domain of the Network Operator could consist of an ME called "Operator A ME" and "Operator B ME". Similarly, if the NNI between Network Operators requires management, an ME called "NNI ME" appears. The focus of the Service OAM framework is on the UNI ME, the EVC ME, and the subscriber ME associated with a service. Figure 7-4 displays a point-to-point maintenance configuration example at the Ethernet layer. The following section describes all of the components in this graphic.

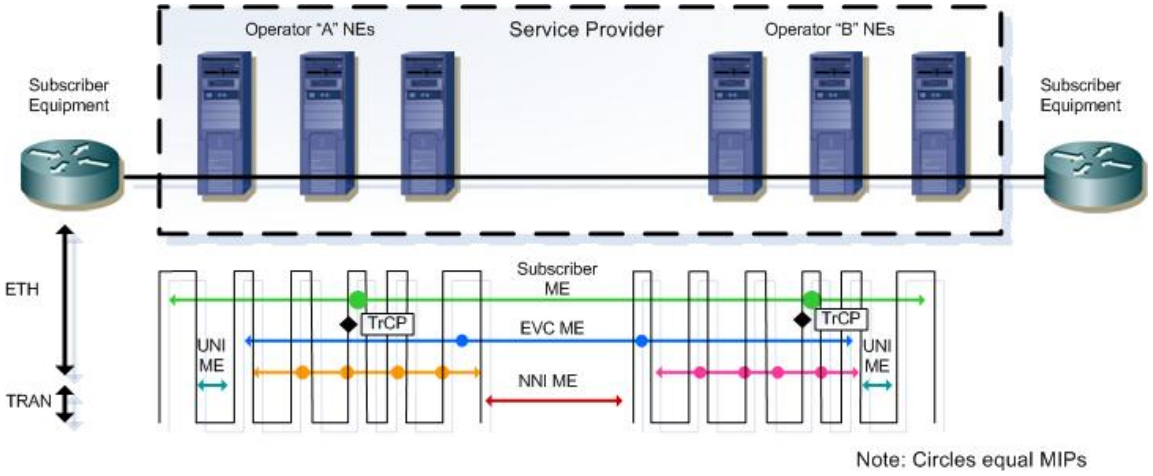


Figure 7-4: Point-to-point Maintenance Entity

Maintenance Entity Group (MEG)

A Maintenance Entity Group (MEG) includes any ME that belongs to the same service within the same OAM domain. A MEG contains a single ME for a point-to-point EVC and contains multiple MEs for a multipoint-to-multipoint EVC that has multiple UNIs.

Maintenance Entity Group Endpoint (MEP)

In a point-to-point EVC, there are two MEPs—one on each endpoint of the ME. In a multipoint-to-multipoint EVC, where there are multiple UNIs, there are multiple MEPs—again, one on each endpoint.

Maintenance Entity Group Intermediate Point (MIP)

MEG Intermediate Point (MIP) is a provisioned OAM reference point, which is capable of reacting to diagnostic OAM frames initiated by MEPs. A MIP does not initiate proactive or diagnostic OAM frames. The number of MIPs in a point-to-point EVC or Multipoint-to-Multipoint EVC depends on the specific deployment.

Traffic Conditioning Point (TrCP)

A Traffic Conditioning Point (TrCP) corresponds to the Ethernet Subscriber Conditioning Function (ESCG). Traffic conditioning might occur at the UNI-C/UNI-N, as well as at other locations in the network. For example, Service OAM occurs between peer MEP instances of an ME. From the perspective of the network, traffic conditioning performed at TrCPs might occur before or after a given MEP and within the same NE as the MEP or in another NE. As a result, OAM frames generated by a given MEP may or may not be subject to traffic conditioning.

Maintenance Entity Group Level

A Maintenance Entity Group (MEG) Level helps distinguish between OAM frames belonging to different nested MEs. MEs that belong to the same MEG share a common MEG level. Ethernet OAM identifies eight MEG Levels. When subscribers, service providers, and network operators share the MEG Levels space, allocation of MEG Levels can negotiate among the different roles involved. Default allocation of MEG Levels includes the following:

- Service OAM frames for a Subscriber ME use MEG Level 7, 6 or 5
- Service OAM frames for an EVC ME use MEG Level 3 or 4 as EVC ME belongs to a SOAM Domain
- Operator MEs use MEG Levels 2, 1, or 0.
- The MEG Levels used for UNI ME and NNI ME defaults to zero (0).

Note: The default allocation of MEG Level space between Subscribers, Service Providers, and Operators could change based on mutual agreement among the three entities.

Maintenance Entity Group Class of Service (CoS)

The Maintenance Entity Group CoS (class of service) represents one or more priorities associated with the OAM frames for a given ME. Each ME within a MEG shares a common CoS profile. An EVC ME can associate with OAM frames with multiple priorities, since an EVC can associate with service frames with different CoS levels.

Service OAM Requirements

The following section describes the Service OAM requirements.

Discovery

Network equipment with Service OAM capability uses discovery to learn information (for example, MAC addresses, etc.) about other network equipment with Service OAM capability, so that the discovered network equipment can exchange OAM frames. For EVCs, discovery allows network equipment with Service OAM capability to learn about other network equipment with Service OAM capabilities that support the same EVCs. Network equipment must reside at the edges of an OAM domain within which the discovery occurs.

Connectivity

A Maintenance Entity can have the following connectivity status values:

- Active: This value implies that two or more MEPs can exchange Service OAM frames in both directions.

- Not Active: This value implies that the connectivity status is not active between two or more MEPs, so they cannot exchange Service OAM frames.

A Multipoint-to-Multipoint MEG can have the following connectivity status values:

- Active: This value implies that each ME in the MEG has a connectivity status of active.
- Not Active: This value implies that each ME in the MEG has a connectivity status of active.
- Partially Active: This value implies that there exists at least one active and one inactive ME in the MEG.

SOAM monitors the connectivity status of an ME and MEG to detect changes in the connectivity status within a time interval. Typically, this configurable time interval is more than the network restoration time, which depends on the MEN technology. For example, if bridging technology is the basis for a MEN and the network restoration uses xSTP, then the configurable time-interval for connectivity status monitoring of an ME or a MEG must be more than the xSTP restoration time-interval. In addition, when a connectivity status is inactive or partially active, Service OAM can verify and localize the fault. This reduces operating costs by minimizing operational repair times and resources.

SOAM can verify the existence of a connectivity fault or localize a fault inside a service provider OAM domain. Localization helps identify the MEP and MIP (or pairs of MIPs) in the service provider OAM domain in which the EVC connectivity faults exist. SOAM frames checking connectivity transmit at the highest priority permissible for service frames. This ensures that the system is less likely to discard service OAM frames (in comparison to other service frames) when there is network congestion.

The MEP connectivity status for a MEP in a multipoint-to-multipoint MEG can have one the following values:

- Fully Connected: This value implies that all MEs to which the MEP belongs to are active.
- Isolated: This value implies that all MEs to which the MEP belongs to are inactive.
- Partially Connected: This value implies that, among all MEs to which the MEP belongs to, at least one is active and one is inactive.

Frame Loss Ratio Performance

Frame loss ratio (FLR) performance is the percentage of lost frames within the MEN. It applies to all service frames with the level of bandwidth profile conformance determined to be “Green.” FLR associates with a particular CoS instance on a point-to-point EVC that arrives at the UNI during a defined time interval. For a point-to-point EVC, estimating FLR performance is dependent on the ability to measure the frame loss between MEPs of an EVC ME during a defined time interval. Statistics collected at the TrCP points are the basis of the measurements, which determine green, yellow and red service frames.

Frame Delay Performance

Frame Delay is the time required to transmit a Service Frame from source UNI to destination UNI across the MEN. Frame Delay performance for a particular CoS instance on a Point-to-Point EVC measures the delays experienced by different Service Frames belonging to the same CoS instance. Frame Delay performance for a CoS instance on a Point-to-Point EVC during a specific time interval is defined as P-Percentile of the delay for all Service Frames with the level of Bandwidth Profile conformance determined to be Green that were successfully delivered between the UNI pairs during a specific time interval.

For a point-to-point EVC, estimating Frame Delay performance depends on the ability to measure the Frame Delay experienced by the Green Service Frames that belong to a particular CoS instance, which exists between the UNI pairs on a Point-to-Point EVC. If the Service OAM frames receive the same treatment as Green Service Frames between the MEPs of a Point-to-Point EVC ME, these measurements can be approximated by the Frame Delay experienced by the Service OAM frames that belong to the CoS instance.Frame Delay can be of two types:

- One-way Frame Delay: This type of frame delay characterizes the various applications and services (for example, broadcast applications). Its measurement generally requires synchronization of clocks between the two participating network devices.

- Two-way Frame Delay: In most cases, this type of frame delay is the most sufficient metric, since it is the one that influences the quality of applications (for example, the length of silence in IP-phone calls).

Frame Delay Variation Performance

Frame Delay Variation (FDV) is the difference in delay of two Service Frames. The performance of FDV for a specific CoS instance on a Point-to-Point EVC is a measure of the variation in the delays experienced by the different Service Frames that belong to the same CoS instance. In addition, the performance of FDV applies to all successfully delivered service frames with a level of bandwidth profile conformance, which is determined to be Green for a particular CoS instance on a Point-to-Point EVC for a defined time interval.

Service OAM measures the difference between the two-way frame delay estimates of a pair of service frames with the level of bandwidth profile conformance that is determined to be Green and associated with a specific CoS instance between the UNIs of a point-to-point EVC. Likewise, Service OAM measures the difference between the one-way frame delay estimates of a pair of service frames with the level of bandwidth profile conformance, which are determined to be Green and associated with a particular CoS instance between the UNIs of a point-to-point EVC.

Service OAM Transparency

Service OAM frames that belong to an OAM domain originate and terminate within that OAM domain. By default, security implies that an OAM domain must be capable of filtering Service OAM frames. Filtering for Service OAM frames prevents them from leaking outside a defined OAM domain. For hierarchical OAM Domains, either the system discards (when a Service OAM frames belong to same or lower-level OAM Domains) or transparently passes (when a Service OAM frame belongs to higher-level OAM Domain) any Service OAM frames from outside its OAM domain.

TRAN Layer Independence

The TRAN Layer is independent of the ETH Layer, so the TRAN Layer may include its own OAM capabilities. However, when the system detects a fault in the TRAN Layer, it communicates the fault to the ETH Layer. A fault in TRAN Layer should not cause multiple alarms in the ETH Layer, which then results in unnecessary corrective actions taken in the ETH Layer when a user can restore a fault in the TRAN Layer. As a result, the Service OAM is independent of the TRAN Layer OAM, so it allows interworking with TRAN Layer OAM to deliver of fault notification.

Software Design

The following sections describe the software design of Service OAM.

Discover Service-aware Network Equipment (NE)

By default, Service OAM can learn Remote Maintenance Entity Group Endpoints (RMEP) from Continuity Check Messages (CCMs) in the Maintenance Association (MA), even if the user did not configure them as RMEP. Users can utilize a CLI command to view details of the RMEP. However, a learnt RMEP may not appear when a user issues this command. Thus, users must configure an RMEP on the system to converge an RMEP with other configured RMEPs.

Localize Connectivity Faults

Service OAM can localize a connectivity fault within a Service Provider OAM domain, including support for link trace messaging. Service OAM supports Link Trace messaging. However, if a forwarding database (FDB) entry is missing a targeted MAC address (for example, when a MAC address ages out on an intermediate nodes in a bridging environment), then the link trace message does not elicit a link trace reply. Service OAM mitigates this issue. When a fault is detected, if a link trace message is issued through the CLI within the window of MAC address aging (for example, if the default MAC address aging timeout is set to 300 seconds), then the link trace responses are received from all configured intermediate MIPs and the fault is localized.

Monitor the Connectivity Status of a MEP

Service OAM monitors the MEP connectivity status in a multipoint-to-multipoint MEG. The MEP status corresponds to the connectivity status of the MA (or MEG) that the MEP belongs to currently. For example:

- If the MA is Active, then all the MEPS in the MA are Fully Connected.
- If the MA is Partially Active, then all the MEPS in the MA are Partially Connected.
- If the MA is Not Active, then all the MEPS in the MA are Isolated.

In addition, remote MEPs are associated with a MA and not with a particular MEP. For example:

- MEP fully connects when all configured RMEP in the MA converge.
- MEP partial connects when only some RMEP in a MA converge.
- MEP isolates when none of the RMEP in the MA converge.

Therefore, the MEP inherits the connectivity status from the MA. Typically, a user does not want to have MEPs with a different number of RMEPs configured for each MEP in an MA. That would imply that the MA logically divides into multiple logical sub-MAs, each composed of MEPs related to a different set of RMEPs. This is not how an MA should represent in a network per the standard.

Path Followed by Service OAM Frames

Both Service OAM frames and service frames follow the same path across the MEN as service frames in an EVC (that is, Service VLAN). CFM frames originating from the customer edge (CE) are able to traverse a Provider Bridge (PB) network and service frames from the customer equipment are able to traverse such a network. In fact, even in subscriber MA (or MEG) in a PB network, the Service OAM frames traverse the same path as service frames within the network.

CHAPTER 8 Ethernet Protection Switching

ZebOS-XP supports point-to-point Ethernet path protection on a per-VLAN basis, in accordance with the requirements of the ITU-T G.8031 recommendation. A single physical port can have both protected VLANs and non-protected VLANs. (Non-protected VLANs can be managed with a spanning tree protocol.) This allows the same port to have some VLANs under spanning tree control and others under Automated Protection Switching (APS) control.

Features

- Ethernet Protection Switching (EPS) is applicable to Point-to-Point VLAN-based Ethernet Subnetwork Connections (SNC) that provide connectivity between two Ethernet flow points in a flow domain. VLAN IDs (VIDs) are used to identify Point-to-Point VLAN-based Ethernet SNCs in Ethernet links.
- All impaired traffic in a protected domain is protected from a failure on a single working entity.
- Periodic monitoring of the Ethernet layer connectivity of the working transport entity and the protection transport entity is managed by the Connectivity Check Messaging (CCM) procedures defined in ITU-T Y.1731.
- Both individual (uni-directional) and group (bi-directional) protection switching are supported.
- Network operator options include revertive and non-revertive switching.
- Mismatches between the bridge or selector positions of the near end and the far end are detected and reported.
- Bridge/selector mismatches for a local network element are detected and reported. They are cleared by the user.
- Lockout of Protection (LoP), force switch, manual switch commands, and other operator requests are supported.
- Prioritized protection between Signal Fail (SF) and operator requests is supported.
- A provisionable, generic hold-off function is available to delay the beginning of protection-switching actions.
- Protection switching is agnostic to the monitoring method used, as long as it can receive OK or SF information for the transport entity of each point-to-point linear Ethernet link.
- Both software-based,= and hardware-assisted working and protection path switching are supported.
- Transfer time is less than 50 ms when hardware-assisted APS is supported on the target platform.

Architecture

NSM supports Ethernet protection bridge links and VLAN- related configuration. NSM maintains centralized bridge, VLAN, and interface configuration information that is distributed to the ZebOS-XP ONMD (OAM Network Management Daemon) module.The ONMD module supports the protection FSM and APS messaging functions. The ONMD module supports CCM and APS messaging, and the state machines that are driven by Signal Failure (SF), and other events to realize the protection behavior specified in the ITU-T G.8031 recommendation. Figure 8-1 on page 198 shows the relationships among entities in the Layer 2 switching environment.

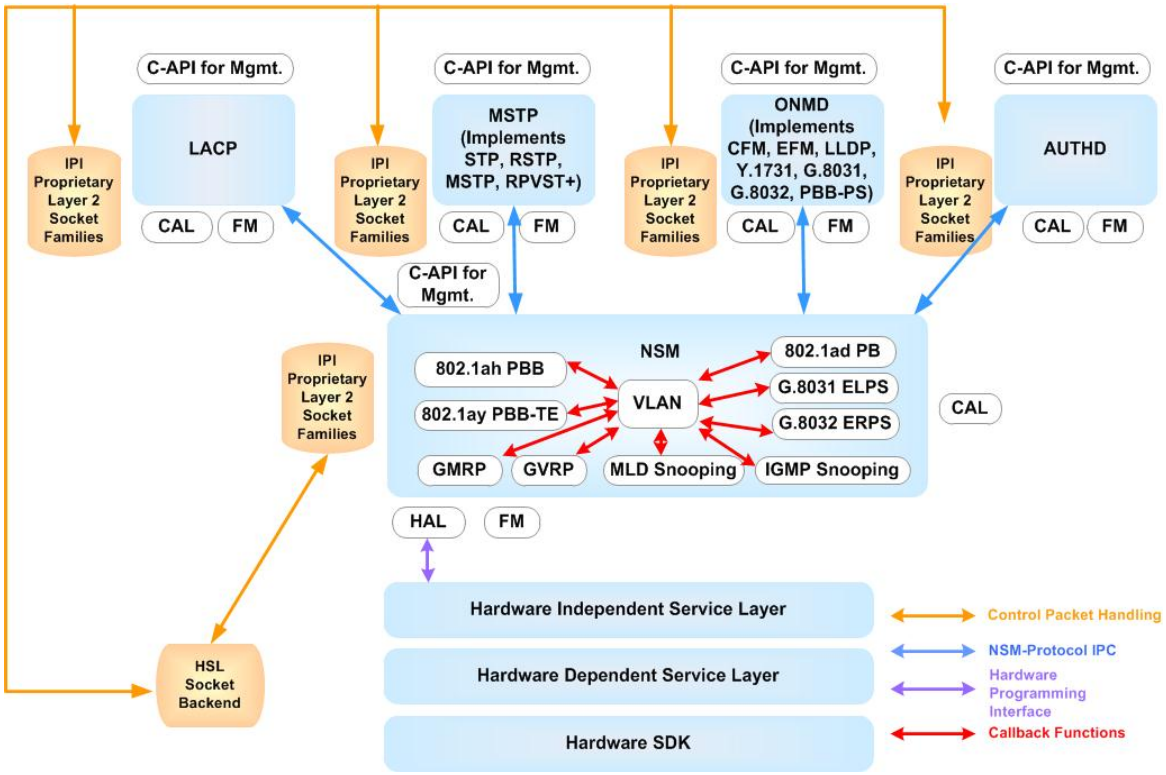


Figure 8-1: EPS in Layer 2 architecture

Switching Architectures

The following switching architectures are managed in the ZebOS-XP implementation of Ethernet Protection Switching.

- 1+1 Unidirectional Non Revertive
- 1+1 Unidirectional Revertive
- 1+1 Bi-directional Non Revertive
- 1+1 Bi-direction Revertive
- 1:1 Bi-directional Non-Revertive
- 1:1 Bi-directional Revertive

Note: All related code for ITU-T G.8031 protection switching is wrapped in the HAVE_EPS_G8031 definition.

Finite State Machines

The following protection switching and recovery FSM are supported.

State Machines	Number of States	Number of FSM Events
For Local Requests		
1:1, bi-directional, revertive mode	9	10
1:1, bi-directional, non-revertive mode	10	9
1+1, bi-directional, revertive mode	9	10

1+1, bi-directional, non-revertive mode	10	9
1+1, uni-directional, revertive mode	7	10
1+1, uni-directional, non-revertive mode	7	9
1+1, uni-directional, revertive mode, no APS messaging	4	3
1+1, uni-directional, non-revertive mode, no APS messaging	4	3
For Far End Requests		
1:1, bi-directional, revertive mode	9	9
1:1, bi-directional, non-revertive mode	10	10
1+1, bi-directional, revertive mode	9	9
1+1, bi-directional, non-revertive mode	10	10

Protection Switching

The ITU-T G.8031 recommendation defines linear (Point-to-Point), 1+1 (One-plus-One), and 1:1 (One-to-One) protection switching (PS) architectures. Linear 1+1 protection switching operates with either uni-directional or bi-directional switching. Linear 1:1 protection switching operates with bi-directional switching. The table below shows the possible protection modes.

	Uni-directional	**Bi-directional**
1+1	√ With or without APS messaging	√ With APS messaging
1:1		√ With APS messaging

With a permanent bridge at the head end (source) and no need to coordinate selector positions at the two ends, the tail end (sink) selector can be driven entirely by defect management, for example, when Connectivity Check Messages (CCM) are not received on the working entity. A tail-end selector can also be managed with operator commands. For example, an operator can execute a Manual Switch. In a particular direction of transmission, the head end (source) of the protected entity is capable of performing a bridge function, and places a copy of a normal traffic signal onto a protection transport entity when required. The tail end (sink) performs a selector function, and is capable of selecting a normal traffic signal, either from its usual working transport entity or from a protection transport entity. In 1+1 bi-directional switching, both directions act as bridge and selector.

Linear 1+1

In the linear 1+1 protection switching architecture, a protection transport entity is dedicated to each working transport entity. Normal traffic is copied and fed to both working and protection transport entities with a permanent bridge at the head end of the protected domain. Traffic on working and protection transport entities is transmitted simultaneously to the tail end of the protected domain, where a selection between the working and protection transport entities is made based on some predetermined criteria, such as server defect identification. Although selection is made only at the tail end of the protected domain in linear 1+1 protection switching architecture, bi-directional 1+1 protection switching must have the APS coordination protocol so that selectors for both directions select the same entity. However, uni-directional 1+1 protection switching does not need APS coordination protocol.

Linear 1:1

In the linear 1:1 protection switching architecture, the protection transport entity is dedicated to the working transport entity. However, the normal traffic is transported either on the working transport entity or on the protection transport entity using a selector bridge at the source of the protected domain. The selector at the sink of the protected domain selects the entity that carries the traffic. Since source and sink need to be coordinated to ensure that the selector bridge at the source and the selector at the sink select the same entity, APS coordination protocol is necessary.

Design

ZebOS-XP EPS depends upon architecture types. Each architecture type has its own finite state machine associated with it. Each state machine responds to the events that occur locally and to the APS messages received from far end. Events that can occur locally include the following:

- Lock of Protection (LoP)
- Forced Switch (FS)
- Manual Switch (MS)
- Exercise (EXR)
- Working Recovers from SF (w.SF)
- Protection Recovers from SF (p.SF)
- Signal Fail on Working
- Signal Fail on Protection
- Clear
- Wait-to-Restore (WTR) timer expires

Mismatches or Failure of Protocol Detection

With various options for provisioning protection groups, there are possibilities for mismatches at either end. Not all provisioning mismatches can be conveyed and detected by information passed through the APS protocol. Wherever applicable, both sides can adapt operations to an internetwork despite mismatches. For example, an entity provisioned for bi-directional switching can fall back to uni-directional switching to allow internetworking. An entity provisioned for 1+1 switching with APS communication can fall back to 1+1 uni-directional switching without APS communication. The operator is informed of the provisioning mismatch, but a level of protection is still provided by the equipment. Failure of protocol defects are classified as follows:

- Fully incompatible provisioning (B-bit mismatch)
  - Entry criteria is “Reception of one APS frame with an incompatible B-bit value”
- Working Protection Configuration Mismatch
  - Reception of one APS frame from the working transport entity
- Protection Switching Incomplete
  - Entry criteria is "If the transmitted ‘Requested Signal’ and received ‘Bridged Signal’ do not match for a period of 50 ms or longer”

EPS Messaging and Local Request Processing

The following is the logic for Local Request Processing for EPS:

1. In a local network element, one or more local protection switching requests (Local Requests) is active.
2. Local Priority Logic determines which of these requests is the top priority, using the order of priority.

3. Top Priority Local Request information is passed on to the Global Priority Logic.
4. Local network element receives APS-Specific Information from the network element at the far end.
5. Received APS-specific information is subjected to a validity check.
6. Information in the received Request/State Byte Far End Request is then passed to Global Priority Logic.

Automated Protection Switching

The Automated Protection Switching strategy involves reserving a protection channel (dedicated or shared) with the same capacity as the channel or facility to be protected.

CCM and APS Messaging

The section describes the messaging model used in the ITU-T G.8031 recommendation for CCM and APS messages.

APS Protocol

APS information is carried in an APS PDU, one of a group of Ethernet OAM PDUs. APS-specific information is transmitted in certain fields of the APS PDU. An APS PDU is identified by a unique Ethernet OAM OpCode (39).

APS PDU. A destination MAC for an APS PDU is dependent upon receipt of a CFM destination MAC address. If a . CFM destination MAC is multicast, APS frames are also transported as multicast. In other words, the same multicast or unicast type selected for CCM is extended to APS. The destAddr is the same for APS as it is for CCM (multicast or unicast Remote MEP MAC).

1-Phase APS

The ITU-T G8031 recommendation defines a 1-phase APS protocol. 1-phase APS does not wait for acknowledgements before changing the bridge/selector state. It sends a message to the far end, then changes its bridge/selector. The tail end signals a Bridge Request to the head end and immediately operates its selector; this is known as a blind switch. The head end operates its bridge upon reception of Bridge Request.

Resources

The ZebOS-XP resources used in APS include the following:

- Timers
- Wait-to-restore timer
- Hold-off timer
- Timer required to send three consecutive APS frames in an interval of 3ms. As soon as a state change takes place, the timer is triggered and sends 3 consecrative APS PDU with required Request/State information.
- A timer for sending consecutive NRs in 5ms intervals

Connectivity Fault Management in APS

APS messages are carried over to ONMD messages. The ONMD module maintains an opcode for APS-specific messages. CFM message parsing and validation is done in the CFM module. ZebOS-XP CFM infrastructure handles APS packet reception and transmission. After validating ONMD header information, messages with an APS opcode are passed to the EPS module. EPS provides a hook for receiving APS packets; upon receiving APS PDU, CFM invokes the hook.

Since EPS works on point-to-point links, traffic flowing across the links may belong to multiple VLANs. Because MEP configuration is done on per-VLAN basis, *n* number of CCMs could start flowing from one end to the other. In order to

prevent this, a primary VID is identified for the entire link, and CCMs belonging to same VID flow from end to end. When an LOC takes place on the primary VID, traffic flowing on all VLANs is switched.

Finite State Machines

As per the ITU-T G8031 recommendation, the maximum states possible is 18. State transitions take place based on incoming events. Incoming events are classified as local events or far-end events. The same event can be triggered from both from local (near end) and from the far end. Even though the event is the same, depending upon its origin processing differs. Two function pointers are registered against each state, one of which is triggered for local requests, and one of which is triggered for far end requests.

Throughout the FSM design, the current state of a protection group is specified as `U_U` or `O_O`, `U_O`. For example, when a protection group is in the `NR_U_U` state, `U_U` is derived from Requested Signal and Bridged Signal. Requested Signal signifies on which entity, working or protection, traffic is flowing, based on the request-type of APS. If traffic is flowing from the Working Entity, the requested signal is NULL, otherwise the requested signal is NORMAL. A Bridged signal signifies whether the bridge is permanent or not. In the case of a Permanent Bridge, the Bridged Signal is considered Normal. In the case of No Permanent Bridge, the Bridged Signal is considered to be NULL or Normal, based on the entity on which traffic is flowing.

Note: In the 1:1 architecture, it is always assumed that there is no permanent bridge.

`NR_U_U` can be read as follows:

- The current state of the protection group is NR. Traffic is flowing thru the working entity and bridging is taking place on the working entity.

Incoming APS packets are initially received by CFM. After all sanity checks are performed, the APS PDU is passed to EPS. To transmit APS frames to the far end, EPS builds an ONMD header, the Destination MAC is used as the RMEP MAC address, and the cfm_mac_relay() function is employed to send the APD PDU to the far end. When an APS message is received, validations are done, then the APS_ID is retrieved using the VLAN_ID in the packet. From the APS_ID, the protection switching structure is retrieved. The current APS state is determined using the protection switching structure and invoking the appropriate FSM function. The protection group structure and incoming event information are passed as arguments. State is changed and an NR with requested signal and bridged signal is sent.

Using the APS_ID, the protection group structure is retrieved. Then the current state is retrieved and the appropriate function is invoked by passing protection group structure and local event information. APS PDUs are sent continuously with triggered local event until the highest-priority APS incoming request is received.

Processing Structure

`The structure used to process all states of APS provided below. It is defined as a three-dimensional array:`

G8031FSM[MAX_ARCH_TYPES][MAX_G8031_STATES][G8031_MAX_EVENTS

struct

```
{
   u_int32_t (*fptr)();
   u_int32_t event_in;
} G8031FSM[MAX_G8031_STATES][G8031_MAX_EVENTS] =
{
   /* The first 15 are remote events and from 16 onwards local */
   /* No-Request State */
     {
       /* Remote events start */
      { g8031_far_opo_handle_nr,           G8031_NO_REQUEST                },
```



```
      { g8031_far_opo_not_applicable,       G8031_DO_NOT_REVERT              },
      { g8031_far_opo_no_state_change,      G8031_REVERSE_REQUEST            },
      { g8031_far_opo_invalid_event  ,      3                                },
      { g8031_far_opo_action_exer,          G8031_EXERCISE                   },
      { g8031_far_opo_not_applicable,       G8031_WAIT_TO_RESTORE            },
      { g8031_far_opo_invalid_event   ,     6                                },
      { g8031_far_opo_invalid_event   ,     7                                },
      { g8031_far_opo_move_to_nr_o_o,       G8031_MANUAL_SWITCH              },
      { g8031_far_opo_no_not_handled,       G8031_SIGNAL_DEGRAGE             },
      { g8031_far_opo_invalid_event   ,     10                               }
      { g8031_far_opo_move_to_nr_o_o ,      G8031_SIGNAL_FAIL_FOR_WORKING    }
      { g8031_far_opo_move_to_nr_o_o ,      G8031_FORCED_SWITCH              },
      { g8031_far_opo_no_state_change ,     G8031_SIGNAL_FAIL_FOR_PROTECTION },
      { g8031_far_opo_no_state_change ,     G8031_LOCKOUT_OF_PROTECTION      },
        /* local events start */
      { g8031_near_opo_move_to_lop,         G8031_LOP                        },
      { g8031_near_opo_move_to_fs ,         G8031_FS                         },
      { g8031_near_opo_move_to_sf_w         G8031_SF_W                       },
      { g8031_near_opo_handle_w_recover,    G8031_W_RECOVERS_FROM_SF         },
      { g8031_near_opo_move_to_sf_p         G8031_SF_P                       },
      { g8031_near_opo_ignore,              G8031_P_RECOVERS_FROM_SF         },
      { g8031_near_opo_move_to_ms,          G8031_MS                         },
      { g8031_near_opo_not_applicable,      G8031_CLEAR                      },
      { g8031_near_opo_move_to_exer,        G8031_EXER                       },
      { g8031_near_opo_not_applicable,      G8031_WTR_EXPIRES                }
    }, /* State End */
}
```

MAX_ARCH_TYPES represents the architecture types supported by G.8031.

MAX_G8031_STATES represents the maximum number of possible states that G.8031FSM can be in. Although the maximum possible states is represented in 4 bits, the maximum states that are possible are represented in 11 bits. For purpose of simplicity, it is assumed that the maximum possible states are 16, and that processing of incoming events only takes place for valid states.

G8031_MAX_EVENTS represents the maximum events that are possible, presently 27. The following list contains additional events other than the states defined in G.8031:

enum g8031_local_events

```
{
   G8031_LOP = 16,
   G8031_FS,
   G8031_SF_W,
   G8031_W_RECOVERS_FROM_SF,
   G8031_SF_P,
   G8031_P_RECOVERS_FROM_SF,
   G8031_MS,
   G8031_CLEAR,
   G8031_EXER,
   G8031_WTR_EXPIRES,
   G8031_LOCAL_FREEZE,
   G8031_LOCAL_LOCKOUT,
};
```

Command API

The functions in this section are called by the commands in the *Carrier Ethernet Command Line Interface Reference Guide*.

Function	Description
delete_g8031_protection_group	Deletes a protection group
g8031_cfm_association	Associates CFM to EPS
g8031_cfm_de_association	De-associates a protection group with CFM
g8031_configure_pg_timer	Sets the wait-to-restore or hold-off timer
g8031_exercise_create_fn	Tests whether APS is operating correctly
g8031_exercise_delete_fn	Clears the exercise command
g8031_force_switch_create_fn	Switches traffic between the EPS working and protection switch paths
g8031_force_switch_delete_fn	Restores the working and protection switching paths to their original states
g8031_handle_local_command	Gives a local (near-end) command.
g8031_lockout_state_create_fn	Locks out switching to the protection path under all circumstances
g8031_lockout_state_delete_fn	Clears a lock out to the protection path.
g8031_manual_switch_create_fn	Manually switches the working and the protection paths
g8031_manual_switch_delete_fn	Restores the working and the protection paths to their original states
g8031_reset_mode_param	Resets mode parameters
g8031_unconfigure_pg_timer	Clears the wait-to-restore or hold-off timer
initialize_g8031_protection_group	Creates a protection group and initializes it with default values
show_bridge_eps_gp	Gets information about all EPS groups on a bridge
show_bridge_eps_gp	Gets information about a specific EPS group on a bridge

Include File

To use the functions in this chapter, you need to include `onmd/cfm/g8031/g8031_api.h`.

delete_g8031_protection_group

This function deletes a protection group from a bridge.

Syntax

```
enum g8031_return_code
delete_g8031_protection_group (u_int16_t eps_id,
```

```
                                    u_int8_t * br_name)
```

Input Parameters

`eps_id`	Protection group ID
`br_name`	Name of the bridge

Output Parameters

None

Return Values

G8031_BRIDGE_NOT_FOUND when the bridge is not found

G8031_PROTECTION_GROUP_NOT_EXISTS when the protection group does not exist

G8031_FAILURE when the function fails

G8031_SUCCESS when the function succeeds

g8031_cfm_association

This function associates CFM to EPS a CFM by providing an MD name, MA name and bridge name.

Syntax

```
enum g8031_return_code
g8031_cfm_association (u_int8_t * md_name,
                       u_int8_t * ma_name,
                       u_int8_t * br_name,
                       u_int16_t eps_id)
```

Input Parameters

`md_name`	Maintenance domain name
`ma_name`	Maintenance association name
`br_name`	Bridge name
`eps_id`	Protection group ID <1-4094>

Output Parameters

None

Return Values

G8031_BRIDGE_NOT_FOUND when the bridge is not found

G8031_PROTECTION_GROUP_NOT_EXISTS when the protection group does not exist

G8031_CFM_MD_NOTFOUND when the MD is not found

G8031_CFM_MA_NOTFOUND when the MA is not found

G8031_CFM_MEP_NOTFOUND when the MEP is not found

G8031_PRIMARY_VID_OF_MA_PG_NOT_MATCHING when the protection group primary VID does not match the MA primary VID

G8031_ONMD_BRIDGE_PORT_WORKING_NOT_FOUND when the protection working onmd bridge port is not found

G8031_CFM_MEP_NOT_ENABLED_FOR_UNICAST when the MEP has not been enabled for unicast CCM

G8031_ONMD_BRIDGE_PORT_PROTECTION_NOT_FOUND when the protection bridge port is not found

CFM_API_ERR_IF_NOT_FOUND when the interface is not found

G8031_FAILURE when the function fails

G8031_SUCCESS when the function succeeds

g8031_cfm_de_association

This function de-associates a protection group with CFM by providing the MD, MA, and bridge.

Syntax

```
enum g8031_return_code
g8031_cfm_de_association (u_int8_t * md_name,
                          u_int8_t * ma_name,
                          u_int8_t * br_name,
                          u_int16_t eps_id)
```

Input Parameters

`md_name`	Maintenance domain name
`ma_name`	Maintenance association name
`br_name`	Bridge name on which the protection group exists
`eps_id`	Protection group ID <1-4094>

Output Parameters

None

Return Values

G8031_BRIDGE_NOT_FOUND when the bridge is not found

G8031_PROTECTION_GROUP_NOT_EXISTS when the protection group does not exist

G8031_CFM_MD_NOTFOUND when the MD is not found

G8031_CFM_MA_NOTFOUND when the MA is not found

G8031_CFM_MEP_NOTFOUND when the MEP is not found

G8031_PRIMARY_VID_OF_MA_PG_NOT_MATCHING when the protection group primary VID does not match the MA primary VID

G8031_ONMD_BRIDGE_PORT_WORKING_NOT_FOUND when the protection working onmd bridge port is not found

G8031_CFM_MEP_NOT_ENABLED_FOR_UNICAST when the MEP has not been enabled for unicast CCM

G8031_ONMD_BRIDGE_PORT_PROTECTION_NOT_FOUND when the protection bridge port is not found

CFM_API_ERR_IF_NOT_FOUND when the interface is not found

G8031_FAILURE when the function fails

G8031_SUCCESS when the function succeeds

g8031_configure_pg_timer

This function sets the wait-to-restore or hold-off timer for a protection group.

Syntax

```
enum g8031_return_code
g8031_configure_pg_timer (u_int8_t * br_name,
                          u_int16_t eps_id,
                          enum ui_configurable type,
                          u_int32_t timeout)
```

Input Parameters

`br_name`	Name of the bridge on which the EPS exists
`eps_id`	Protection group ID <1-4094>
`type`	Type of timer; one of these values from the `ui_configurable` enum in `onmd/cfm/g8031/g8031_api.h`:
	`WAIT_TO_RESTORE_TIMER`
	`HOLD_OFF_TIMER`
`timeout`	Timeout value <300-720>

Output Parameters

None

Return Values

G8031_PROTECTION_GROUP_NOT_EXISTS when the protection group does not exist

G8031_WTR_NOT_ALLOWED_FOR_NON_REVERTIVE when the trying to configure WTR for a protection group in non-revertive mode

G8031_WTR_ALREADY_RUNNING when a WTR timer is already running

G8031_INVALID_WTR_TIMEOUT when the trying to configure invalid WTR value

G8031_INVALID_WTR_NOT_MULTIPLES_OF_ONE when the WTR is not a multiple of 1 minute

G8031_INVALID_HOLD_OFF when the `timeout` is invalid

G8031_FAILURE when the function fails

G8031_SUCCESS when the function succeeds

g8031_exercise_create_fn

The API is called by the `g8031 exercise` command to test whether APS is operating correctly.

Syntax

```
enum g8031_return_code
g8031_exercise_create_fn (u_int32_t eps_id,
                          u_int8_t * br_name)
```

Input Parameters

`eps_id`	Protection group ID <1-4094>
`br_name`	Name of the bridge on which the protection group exists

Output Parameters

None

Return Values

G8031_BRIDGE_NOT_FOUND when the bridge is not found

G8031_PROTECTION_GROUP_NOT_EXISTS when the protection group does not exist

G8031_PG_NOT_IN_INIT_STATE when the protection group parameters are not set

G8031_EXERCISE_NOT_ALLOWED when the exercise is for unidirectional switching

G8031_PROTECTION_GROUP_IN_LOCAL_LOCKOUT_STATE when the protection group is in local lockout state

G8031_LOWER_PRIO_CMD_NOT_ALLOWED when a lower priority command is not accepted

G8031_FAILURE when the function fails

G8031_SUCCESS when the function succeeds

g8031_exercise_delete_fn

The API is called by the `no` form of the `g8031 exercise` command to clear the exercise command.

Syntax

```
enum g8031_return_code
g8031_exercise_delete_fn (u_int32_t eps_id,
                          u_int8_t * br_name)
```

Input Parameters

`eps_id`	Protection group ID <1-4094>
`br_name`	Name of the bridge on which the protection group exists

Output Parameters

None

Return Values

G8031_BRIDGE_NOT_FOUND when the bridge is not found

G8031_PROTECTION_GROUP_NOT_EXISTS when the protection group does not exist

G8031_FSM_NOT_IN_EXER when not in exercise state

G8031_FAILURE when the function fails

G8031_SUCCESS when the function succeeds

g8031_force_switch_create_fn

This function is called by the `g8031 force` command to voluntarily switch traffic between the EPS working and protection switch paths.

Syntax

```
enum g8031_return_code
g8031_force_switch_create_fn (u_int32_t eps_id,
                              u_int8_t * br_name)
```

Input Parameters

`eps_id`	Protection group ID <1-4094>
`br_name`	Name of the bridge on which the protection group exists

Output Parameters

None

Return Values

G8031_BRIDGE_NOT_FOUND when the bridge is not found

G8031_PROTECTION_GROUP_NOT_EXISTS when the protection group does not exist

G8031_PROTECTION_GROUP_LOCALLY_FREEZED when the protection group is locally frozen

G8031_PROTECTION_GROUP_IN_LOCAL_LOCKOUT_STATE when the protection group is in local lockout state

G8031_LOWER_PRIO_CMD_NOT_ALLOWED when a lower priority command is not accepted

G8031_FORCE_SWITCH_ALREADY_EXISTS when already in forced switched state

G8031_FAILURE when the function fails

G8031_SUCCESS when the function succeeds

g8031_force_switch_delete_fn

This function is called by the `no` form of the `g8031 force` command to restore the working and protection switching paths to their original states.

Syntax

```
enum g8031_return_code
g8031_force_switch_delete_fn (u_int32_t eps_id,
                              u_int8_t * br_name)
```

Input Parameters

`eps_id`	Protection group ID <1-4094>
`br_name`	Name of the bridge on which the protection group exists

Output Parameters

None

Return Values

G8031_BRIDGE_NOT_FOUND when the bridge is not found

G8031_PROTECTION_GROUP_NOT_EXISTS when the protection group does not exist

G8031_FSM_NOT_IN_FORCED_SWITCH_STATE when not in force switched state

G8031_FAILURE when the function fails

G8031_SUCCESS when the function succeeds

g8031_handle_local_command

This function gives a local (near-end) command.

Syntax

```
enum g8031_return_code
g8031_handle_local_command (u_int8_t * br_name,
                            u_int16_t eps_id,
                            u_int8_t   local_cmd)
```

Input Parameters

`br_name`	Bridge name
`eps_id`	Protection group ID <1-4094>
`local_cmd`	Local command; one of these constants from `onmd/cfm/g8031/g8031.h`:

`G8031_LOCAL_FREEZE`

`G8031_LOCAL_LOCKOUT`

`G8031_CLEAR_LOCAL_FREEZ`

`G8031_CLEAR_LOCAL_LOCKOUT`

Output Parameters

None

Return Values

G8031_PROTECTION_GROUP_NOT_EXISTS when the protection group does not exist

G8031_PROTECTION_ALREADY_IN_FREEZED_STATE when the protection group is already in frozen state

G8031_PROTECTION_ALREADY_IN_LOCAL_LOCKEDOUT when the protection group is already in local lockout state

G8031_PROTECTION_GROUP_NOT_IN_FREEZE_STATE when the protection group is not in frozen state

G8031_PROTECTION_GROUP_NOT_IN_LOCAL_LOCKOUT when the protection group is not in local lockout state

G8031_FAILURE when the function fails

G8031_SUCCESS when the function succeeds

g8031_lockout_state_create_fn

This function is called by the `g8031 lockout` command to lock out switching to the protection path under all circumstances.

Syntax

```
enum g8031_return_code
g8031_lockout_state_create_fn (u_int32_t eps_id,
                               u_int8_t * br_name)
```

Input Parameters

`eps_id`	Protection group ID <1-4094>
`br_name`	Name of the bridge

Output Parameters

None

Return Values

G8031_BRIDGE_NOT_FOUND when the bridge is not found

G8031_PROTECTION_GROUP_NOT_EXISTS when the protection group does not exist

GG8031_PG_NOT_IN_INIT_STATE when the protection group parameters are not set

G8031_PROTECTION_GROUP_LOCALLY_FREEZED when the protection group locally frozen

G8031_PROTECTION_GROUP_IN_LOCAL_LOCKOUT_STATE when the protection group is in local lockout state

G8031_LOCKOUT_ALREADY_EXISTS when already in lockout of protection state

G8031_SUCCESS when the function succeeds

g8031_lockout_state_delete_fn

This function is called by `no` form of the `g8031 lockout` command to clear a lockout to the protection path.

Syntax

```
enum g8031_return_code
g8031_lockout_state_delete_fn (u_int32_t eps_id,
                               u_int8_t * br_name)
```

Input Parameters

`eps_id`	Protection group ID <1-4094>
`br_name`	Name of the bridge

Output Parameters

None

Return Values

G8031_BRIDGE_NOT_FOUND when the bridge is not found

G8031_PROTECTION_GROUP_NOT_EXISTS when the protection group does not exist

G8031_FSM_NOT_IN_LOP when not in lockout of protection state

G8031_FAILURE when the function fails

G8031_SUCCESS when the function succeeds

g8031_manual_switch_create_fn

This function is called by the `g8031 manual` command to manually switch the working and the protection paths.

Syntax

```
enum g8031_return_code
g8031_manual_switch_create_fn (u_int32_t eps_id,
                               u_int8_t * br_name)
```

Input Parameters

`eps_id`	Protection group ID <1-4094>
`br_name`	Name of the bridge

Output Parameters

None

Return Values

G8031_BRIDGE_NOT_FOUND when the bridge is not found

G8031_PG_NOT_IN_INIT_STATE when the protection group parameters are not set

G8031_LOWER_PRIO_CMD_NOT_ALLOWED when a lower priority command is not accepted

G8031_PROTECTION_GROUP_LOCALLY_FREEZED when the protection group locally frozen

G8031_PROTECTION_GROUP_IN_LOCAL_LOCKOUT_STATE when the protection group is in local lockout state

G8031_MANUAL_SWITCH_ALREADY_EXISTS when already in manual switched state

G8031_SUCCESS when the function succeeds

g8031_manual_switch_delete_fn

The function is called by `no` form of the `g8031 manual` command to restore the working and the protection paths to their original states.

Syntax

```
enum g8031_return_code
g8031_manual_switch_delete_fn (u_int32_t eps_id,
                               u_int8_t * br_name)
```

Input Parameters

`eps_id`	Protection group ID <1-4094>
`br_name`	Name of the bridge

Output Parameters

None

Return Values

G8031_BRIDGE_NOT_FOUND when the bridge is not found

G8031_PROTECTION_GROUP_NOT_EXISTS when the protection group does not exist

G8031_STATE_NOT_AVAILABLE when the current state is not available

G8031_FSM_NOT_IN_MS when not in manual switch state

G8031_SUCCESS when the function succeeds

g8031_reset_mode_param

This function resets mode parameters for a protection group.

Syntax

```
enum g8031_return_code
g8031_reset_mode_param(u_int16_t eps_id,
    u_int8_t * br_name)
```

Input Parameters

`eps_id`	Protection group ID <1-4094>
`br_name`	Name of the bridge

Output Parameters

None

Return Values

G8031_BRIDGE_NOT_FOUND when the bridge is not found

G8031_PROTECTION_GROUP_NOT_EXISTS when the protection group does not exist

G8031_SUCCESS when the function succeeds

g8031_unconfigure_pg_timer

This function is used to clear the wait-to-restore or hold-off timer for a protection group.

Syntax

```
enum g8031_return_code
g8031_unconfigure_pg_timer (u_int8_t * br_name,
                            u_int16_t eps_id,
                            enum ui_configurable type,
                            u_int32_t timeout)
```

Input Parameters

`br_name`	Bridge name
`eps_id`	Protection group ID <1-4094>
`type`	Type of timer; `WAIT_TO_RESTORE_TIMER` or `HOLD_OFF_TIMER`
`timeout`	Timeout value <300-720>

Output Parameters

None

Return Values

G8031_PROTECTION_GROUP_NOT_EXISTS when the protection group does not exist

G8031_WTR_NOT_ALLOWED_FOR_NON_REVERTIVE when trying to unconfigure WTR when the protection group is configured in non-revertive mode

G8031_INVALID_TIMER_VALUE when `timeout` is invalid

G8031_SUCCESS when the function succeeds

initialize_g8031_protection_group

This function creates a protection group and initializes it with default values.

Syntax

```
enum g8031_return_code
initialize_g8031_protection_group (u_int8_t g8031_br_mode,
                                   u_int8_t revertive,
                                   u_int8_t direction,
                                   u_int8_t *br_name,
                                   u_int16_t eps_id)
```

Input Parameters

`g8031_br_mode`	Ethernet Protection Switching mode: 1 for 1+1 protection switching or 0 for 1:1 protection switching
`revertive`	1 for revertive mode or 0 for non-revertive
`direction`	1 for bidirectional or 0 for unidirectional
`br_name`	Name of the bridge
`eps_id`	Protection group ID <1-4094>

Output Parameters

None

Return Values

G8031_PROTECTION_GROUP_NOT_EXISTS when the protection group does not exist

G8031_PG_ALREADY_INITIALIZED when the protection group is already initialized

G8031_SUCCESS when the function succeeds

show_bridge_eps_gp

This function gets information about a specific EPS group on a bridge.

Syntax

```
void
show_bridge_eps_gp(struct cli *cli,
                   u_int8_t *br_name,
                   uint32_t eps_id)
```

Input Parameters

`br_name`	Name of the bridge
`eps_id`	Protection group ID for which information is requested

Output Parameters

`cli`	Pointer to the CLI structure

Return Values

None

CHAPTER 9 G.8032 (ERPS) Version 2

This chapter describes the ZebOS-XP implementation of ITU-T G.8032 Ethernet Ring Protection Switching (ERPS). “Version 2” refers to ITU-T Recommendation G.8032/Y.1344 as published in February 2012.

Overview

ERPS protects Ethernet traffic in a ring topology by ensuring that no loops are within the ring. Loops are prevented by blocking traffic on either a predetermined link or a failed link. ERPS integrates Ethernet operations, administration, and maintenance (OAM) functions with a simple automatic protection switching (APS) protocol. An Ethernet ring uses normal learning, forwarding, filtering, and flooding mechanisms and a forwarding data base (FDB).

Ring Protection Links

An Ethernet ring consists of multiple ring nodes. Each node is connected to adjacent nodes using two independent ring links. A ring link is bound by two adjacent nodes, each with a port (often called a “ring port”). There must be at least two nodes in a ring.

To avoid loops, an Ethernet ring uses a link called the Ring Protection Link (RPL) upon which traffic is blocked as shown in Figure 9-1.

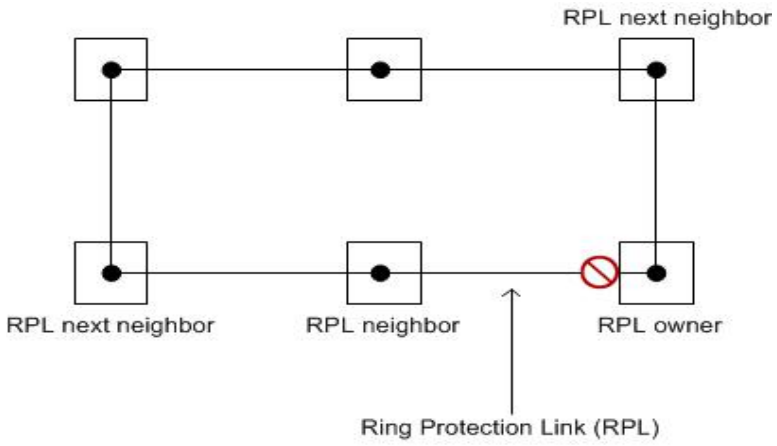


Figure 9-1: G.8032 Ethernet ring

Figure 9-1 also shows these types of nodes:

- RPL owner: Responsible for blocking traffic over the RPL so that no loops are formed. There can be only one RPL owner in a ring.
- RPL neighbor: A node adjacent to the RPL that is responsible for blocking its end of the RPL under normal conditions.
- RPL next neighbor: A node adjacent to an RPL owner node or RPL neighbor node.

R-APS Messages

Nodes on the ring use Ring Automatic Protection Switching (R-APS) messages to coordinate the activities of switching the RPL on and off. Any failure along the ring triggers a signal failure (SF) message in both directions of the nodes adjacent to the failed link, after the nodes have blocked the port facing the failed link. On receiving this message, the RPL owner unblocks the RPL port.

Connectivity Fault Management (CFM) and line status messages are used to detect ring link and node failure.

Revertive Operation

ERPS can be revertive or non-revertive. Revertive means returning the topology to its original state after a failure has been corrected. Non-revertive means the topology does not return to its original state after a failure has been detected.

Interconnected Rings

G.8032 supports a multi-ring/ladder network that consists of Ethernet rings connected by one or more interconnection nodes:

- A major ring constitutes a closed ring controlled by its own ERP instance with its own RPL.
- A sub-ring is connected to a major ring or upper sub-ring via interconnection nodes and does not constitute a closed ring. A sub-ring is controlled by its own ERP instance with its own RPL.

Figure 9-2 shows a basic G.8032 topology with a major ring and an interconnected sub-ring. See ITU-T Recommendation G.8032/Y.1344 for additional examples of topologies.

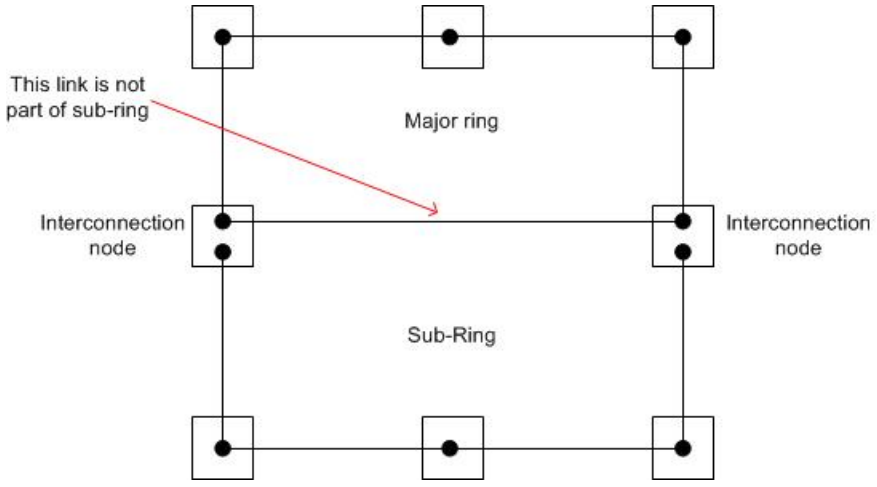


Figure 9-2: Major ring and sub-ring

Virtual and Non-Virtual Channels

A sub-ring does not control or directly transfer R-APS messages over the link between the interconnection nodes which are under the control of the major ring. However, there are two options for the R-APS control channel of a sub-ring:

- A *non-virtual channel*, where R-APS messages terminate at the interconnection nodes, but are not blocked at the sub-ring RPL.
- A *virtual channel*, where R-APS messages are tunneled from one interconnection node to the other interconnection node, but are blocked at the sub-ring RPL.

To direct R-APS traffic on a virtual channel, you specify a dedicated VLAN on the major ring. You need to allocate different VLAN identifiers to differentiate between each R-APS channel within a whole interconnected network.

Figure 9-3 shows an example of a virtual channel.

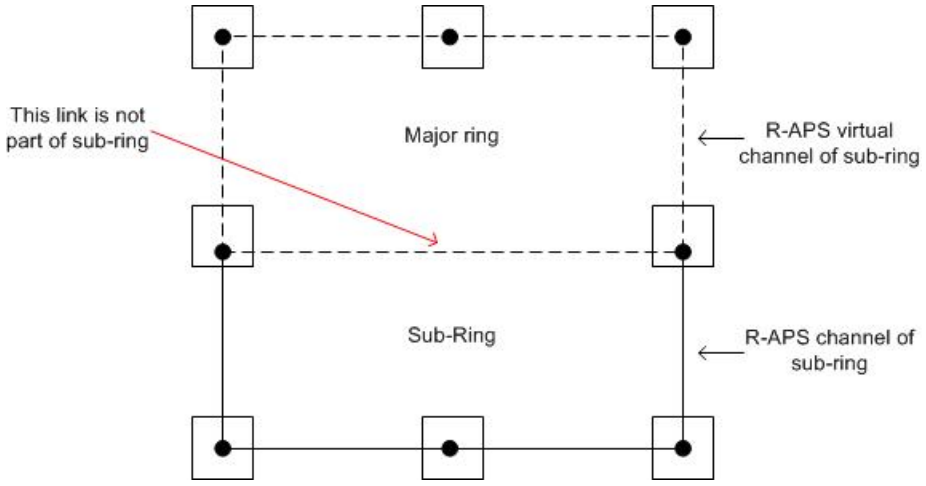


Figure 9-3: Sub-Ring virtual channel

ERPS Instances

An ERPS instance is a logical ring running over a physical ring. An Ethernet ring can support multiple instances, with each instance handling traffic for different groupings of VLANS. Each instance is responsible for the protection of the its VLANs that transport traffic over the physical ring. Each instance is independent of other instances on the physical ring.

A specific VLAN only can be configured under only one instance. A VLAN cannot overlap multiple instances.

Each instance should be configured with its own RPL, RPL owner node, and RPL neighbor node.

Profiles

In ZebOS-XP, an ERPS profile specifies timer values and revertive behavior for an instance. You define profiles independent of instances and then apply a specific profile to an instance.

Administrative Commands

A G.8032 ring supports these administrative commands:

- Force switch (FS): Forcefully block a particular ring port:
  - Effective even if there is an existing signal failure condition
  - Multiple FS commands for a ring are supported
  - Can be used to allow immediate maintenance operations
- Manual switch (MS): Manually block a particular ring port:
  - Ineffective in an existing FS or signal failure condition
  - Overridden by new FS or SF conditions
  - Multiple MS commands cancel all MS commands
- Clear: Cancels an existing FS or MS command on the ring port. The Clear command is used at the RPL owner to clear a non-revertive mode condition.

Timers

The ERP protocol specifies the use of different timers to avoid race conditions and unnecessary switching operations:

- After a signal failure (SF) condition, a Wait-to-Restore (WTR) timer verifies that the signal failure is not intermittent.
- After a force switch or a manual switch command, a Wait-to-Block (WTB) timer verifies that no background condition exists.
- Guard timer: Used by all nodes when changing state to block latent outdated messages from causing unnecessary state changes.
- Hold-off timer: Used by the underlying Ethernet layer to filter out intermittent link faults. Faults are reported to the ring protection mechanism only if this timer expires.

Data Structures

g8032_erps_instance

This data structure in `onmd/cfm/g8032v2/g8032_erps.h` represents an ERPS instance.

Member	Description
bridge	Back pointer to the ERPS bridge
id	Unique identifier for the ERPS instance
erps_name	Configurable string name for the instance
node_id	Highest MAC address from the instance's interfaces
state	Current state of the instance; changed according to state machine
inst_bitmask	Various bits marking TCN, virtual-channel status, DNS bit set, and so on
role	Role of the instance on this machine such as owner, neighbor, and so on
prof	Link to the configuration profile
east	Link to the east interface of the instance
west	Link to the west interface of the instance
phy_ring	Lnk to the physical ring
data_traffic_vlans	List of data traffic VLANS to block/unblock
raps_channel	Link to R-APS channel with information about level and ring identifier
wtr_timer	Thread for wait-to-restore timer
wtb_timer	Thread for wait-to-block timer
holdoff_timer	Thread for hold-off timer
guard_timer	Thread for guard timer
msg_timer_val	Timer value interval for sending R-APS messages
msg_timer	Thread for R-APS message transmission

Member	Description
vc_msg_timer	Thread for R-APS message transmission of the virtual channel through this instance
top_prio_req	Current top priority local request

Definition

```
struct g8032_erps_instance
{
    struct g8032_bridge *bridge;
    u_int32_t id; /* Auto */
    u_char erps_name[G8032_ERPS_NAME_MAX_LEN];
    u_int8_t node_id[ETHER_ADDR_LEN];
    u_int32_t state;
    u_int8_t inst_bitmask;
    enum g8032_instance_role role;
    struct g8032_instance_profile *prof;
    struct g8032_raps_channel_link *east;
    struct g8032_raps_channel_link *west;
    struct g8032_phy_ring *phy_ring;
    struct list data_traffic_vlans; /*g8032_data_traffic*/
    struct g8032_channel_info  *raps_channel;
    /* Timers */
    struct thread *wtr_timer;
    struct thread *wtb_timer;
    struct thread *holdoff_timer;
    struct thread *guard_timer;
    u_int32_t msg_timer_val;  /* msg interval 5 secs */
    struct thread *msg_timer;
    struct thread *vc_msg_timer;
    u_int8_t top_prio_req;
};
```

g8032_instance_profile

This data structure defined in `onmd/cfm/g8032v2/g8032_profile.h` represents a profile.

Member	Description
`prof_name`	Profile name
`wtr_timer_val`	Wait-to-restore in seconds
`wtb_timer_val`	Wait-to-block time in seconds
`guard_timer_val`	Guard time in microseconds
`holdoff_timer_val`	Hold-off time in microseconds

Member	Description
is_revertive	Whether the instance switching is revertive
erps_list	List of instances associated to this profile

Definition

```
struct g8032_instance_profile
{
    u_char prof_name[G8032_ERPS_PROFILE_NAME_MAX_LEN];

    u_int32_t wtr_timer_val;      /* Wait-to-restore in secs */
    u_int32_t wtb_timer_val;      /* Wait-to-block in secs */
    u_int32_t guard_timer_val;    /* guard in microsecs */
    u_int32_t holdoff_timer_val;  /* holdoff in microsecs */

    u_int8_t is_revertive;

    struct list  erps_list;          /* List of associated instances
                                     * to this profile */
};
```

Command API

The functions in this section are called by the commands in the *Carrier Ethernet Command Reference Guide*.

Function	Description
g8032_api_erp_inst_admin_cmd	Configures administrative commands
g8032_api_erp_inst_create	Creates or deletes an ERP instance
g8032_api_erp_inst_set_data_traffic	Adds or removes a VLAN to/from an ERP instance as a data traffic channel
g8032_api_erp_inst_set_ins_type	Makes an ERP instance a major ring or a sub-ring
g8032_api_erp_inst_set_mel	Sets the MEG level to carry in RAPS messages
g8032_api_erp_inst_set_phy_ring	Associates an ERP instance to a physical ring
g8032_api_erp_inst_set_profile	Associates an ERP instance to a profile
g8032_api_erp_inst_set_raps_channel	Adds or removes a VLAN to/from an ERP instance as a RAPS channel
g8032_api_erp_inst_set_ring_id	Sets the ring identifier
g8032_api_erp_inst_set_rpl_role	Sets the RPL role of the ring node
g8032_api_erp_inst_set_version	Sets the version to carry in RAPS messages for an interface

Function	Description
g8032_api_erp_inst_set_virt_channel	Creates or deletes a virtual channel or a non-virtual channel
g8032_api_erp_inst_show	Gets details about an ERP instance
g8032_api_phy_ring_create	Creates or deletes a physical ring
g8032_api_phy_ring_show	Gets details about a physical ring
g8032_api_profile_create	Creates or deletes a profile
g8032_api_profile_set_revertive	Sets the revertive behavior of the ring node
g8032_api_profile_set_timer	Sets a timer
g8032_api_profile_show	Gets details about a profile
g8032_api_tcn_propogation	Enables or disables TCN propagation

Include Files

To call the functions in this chapter, you must include `onmd/cfm/g8032v2/g8032_api.h`.

g8032_api_erp_inst_admin_cmd

This function configures administrative commands.

This function implements the `force-switch|manual-switch` command.

Syntax

```
s_int32_t
g8032_api_erp_inst_admin_cmd (struct g8032_erps_instance *inst,
                              enum g8032_admin_cmd cmd,
                              u_int32_t is_east)
```

Input Parameters

`inst` ERP instance

`cmd` Role; one of these constants from the `g8032_admin_cmd` enum in `onmd/cfm/g8032v2/g8032_api.h`:

`G8032_ADMIN_CMD_FS`
Forcefully block a ring port

`G8032_ADMIN_CMD_MS`
Manually block a ring port

`G8032_ADMIN_CMD_CLEAR`
Cancel a command

`is_east` Whether to apply the command to the east or west interface:

`PAL_TRUE` Apply command to east interface

`PAL_FALSE` Apply command to west interface

Output Parameters

None

Return Values

G8032_ERROR_NONE when the function succeeds

G8032_ERROR_ERPS_INST_NOT_FOUND when `inst` is NULL

g8032_api_erp_inst_create

This function creates or deletes an ERP instance.

This function implements the `g8032 erp-instance` command.

Syntax

```
s_int32_t
g8032_api_erp_inst_create (u_int8_t is_conf,
                           u_char *br_name,
                           u_char *inst_name,
                           struct g8032_erps_instance **inst)
```

Input Parameters

`is_conf`	Whether creating or deleting an ERP instance:
`PAL_TRUE`	Creating an instance
`PAL_FALSE`	Deleting an instance
`br_name`	Bridge name
`inst_name`	Instance name

Output Parameters

`inst`	ERP instance. Specify NULL when deleting an instance.

Return Values

G8032_ERROR_NONE when the function succeeds or when creating an instance and `inst_name` already exists

G8032_ERROR_MAX_ALLOWED_ERPS_INSTANCE_CONFIGURATION when there are 254 existing instances on the bridge

G8032_ERROR_BRIDGE_NOT_FOUND when the bridge is not found

G8032_ERROR_ALLOCATION_FAILED when there is an error allocating memory

G8032_ERROR_ERPS_INST_NOT_FOUND when deleting an instance and `inst_name` is NULL

G8032_ERROR_ERPS_INST_HAVE_ATTACHED_INSTANCES when deleting an instance and `inst_name` has attached instances

G8032_ERROR_UNKNOWN when deleting an instance and both the bridge and instance are NULL

g8032_api_erp_inst_set_data_traffic

This function adds or removes a VLAN to or from an ERP instance as a data traffic channel.

This function implements the `vlan <2-4094> data-traffic` command.

Syntax

```
s_int32_t
g8032_api_erp_inst_set_data_traffic (u_int8_t is_conf,
                                     struct g8032_erps_instance *inst,
                                     u_int16_t vid)
```

Input Parameters

`is_conf`	Whether adding or removing a VLAN:
`PAL_TRUE`	Adding a VLAN
`PAL_FALSE`	Removing a VLAN
`inst`	ERPS instance
`vid`	VLAN identifier

Output Parameters

None

Return Values

G8032_ERROR_NONE when the function succeeds

G8032_ERROR_ERPS_INST_NOT_FOUND when `inst` is NULL

G8032_ERROR_RAPS_INVALID_VID when `vid` is not in the range <1-4094>

G8032_ERROR_ERP_INST_VLAN_NOT_CONFIGURED when `vid` is not associated with the bridge

G8032_ERROR_DATA_TRAFFIC_ALREADY_CONFIGURED when the instance is configured for data traffic

G8032_ERROR_RAPS_CHANNEL_ALREADY_CONFIGURED when `vid` is already associated to an ERPS instance

G8032_ERROR_PHY_RING_NOT_ASSOCIATED_WITH_INSTANCE when a physical ring is not associated with the instance

G8032_ERROR_ALLOCATION_FAILED when memory allocation fails

g8032_api_erp_inst_set_ins_type

This function makes an ERP instance a major ring or a sub-ring.

This function implements the `sub-ring` command.

Syntax

```
s_int32_t
g8032_api_erp_inst_set_ins_type (u_int8_t is_conf,
                                 struct g8032_erps_instance *inst,
                                 u_int8_t is_subring,
                                 u_int8_t block_east)
```

Input Parameters

`is_conf`	Whether making the ERP instance a sub-ring or a major ring:
`PAL_TRUE`	Making a sub-ring
`PAL_FALSE`	Making a major ring
`inst`	ERP instance

is_subring	Reserved for future use
block_east	Whether to block the east or west interface. This parameter is ignored when is_conf is PAL_FALSE.
PAL_TRUE	Block east interface
PAL_FALSE	Block west interface

Output Parameters

None

Return Values

G8032_ERROR_NONE when the function succeeds

G8032_ERROR_ALREADY_SUBRING when the instance is already a sub-ring

G8032_ERROR_CONF_SUBRING_NOT_ALLOWED when the instance is not in the initialize state

G8032_ERROR_UNKNOWN when inst is NULL

G8032_ERROR_ALREADY_NOT_SUBRING when trying to make the instance a major ring when it already is

g8032_api_erp_inst_set_mel

This function sets the maintenance entity group (MEG) level (MEL) to carry in RAPS messages.

This function implements the level command.

Syntax

```
s_int32_t
g8032_api_erp_inst_set_mel (struct g8032_erps_instance *inst, u_int8_t level)
```

Input Parameters

inst	ERPS instance
level	Level

Output Parameters

None

Return Values

G8032_ERROR_NONE when the function succeeds

G8032_ERROR_ERPS_INST_NOT_FOUND when inst is NULL

G8032_ERROR_MEL_INVALID when level is not in the range <0-7>

g8032_api_erp_inst_set_phy_ring

This function associates an ERP instance to a physical ring.

This function implements the physical-ring command.

Syntax

```
s_int32_t
```

```
g8032_api_erp_inst_set_phy_ring (u_int8_t is_conf,
                                 struct g8032_erps_instance *inst,
                                 u_char *ring_name)
```

Input Parameters

`is_conf`	Reserved for future use
`inst`	ERP instance
`ring_name`	Physical ring name

Output Parameters

None

Return Values

G8032_ERROR_NONE when the function succeeds

G8032_ERROR_ERPS_INST_NOT_FOUND when `inst` and `ring_name` are NULL

G8032_ERROR_PHY_RING_NOT_FOUND when the physical ring is not found

g8032_api_erp_inst_set_profile

This function associates an ERP instance to a profile.

This function implements the `profile name` command.

Syntax

```
s_int32_t
g8032_api_erp_inst_set_profile (u_int8_t is_conf,
                                struct g8032_erps_instance *inst,
                                u_char *profile_name)
```

Input Parameters

`is_conf`	Reserved for future use
`inst`	ERP instance
`profile_name`	Profile name

Output Parameters

None

Return Values

G8032_ERROR_NONE when the function succeeds

G8032_ERROR_ERPS_INST_NOT_FOUND when `inst` is NULL

G8032_ERROR_PROFILE_NOT_FOUND when the profile is not found

g8032_api_erp_inst_set_raps_channel

This function adds or removes a VLAN to or from an ERP instance as a RAPS channel.

This function implements the `vlan <2-4094> raps-channel` command.

Syntax

```
s_int32_t
g8032_api_erp_inst_set_raps_channel (u_int8_t is_conf,
                                     struct g8032_erps_instance *inst,
                                     u_int16_t vid)
```

Input Parameters

`is_conf`	Whether adding or removing a VLAN:
`PAL_TRUE`	Adding a VLAN
`PAL_FALSE`	Removing a VLAN
`inst`	ERPS instance
`vid`	VLAN identifier

Output Parameters

None

Return Values

G8032_ERROR_NONE when the function succeeds

G8032_ERROR_ERPS_INST_NOT_FOUND when `inst` is NULL

G8032_ERROR_RAPS_INVALID_CHANNEL_TYPE when there is an internal error

G8032_ERROR_RAPS_INVALID_CHANNEL_VID when `vid` is not in the range <1-4094>

G8032_ERROR_ERP_INST_VLAN_NOT_CONFIGURED when `vid` is not associated with the bridge

G8032_ERROR_DATA_TRAFFIC_ALREADY_CONFIGURED when the instance is configured for data traffic

G8032_ERROR_RAPS_CHANNEL_ALREADY_CONFIGURED when `vid` is already associated to an ERPS instance

G8032_ERROR_PHY_RING_NOT_ASSOCIATED_WITH_INSTANCE when a physical ring is not associated with the instance

G8032_ERROR_RAPS_CHANNEL_UNCONFIGE_NOT_ALLOWED when removing and `inst` has attached instances

G8032_ERROR_RAPS_CHANNEL_NOT_FOUND when the channel is not configured

g8032_api_erp_inst_set_ring_id

This function sets the ring identifier.

This function implements the `ring-id` command.

Syntax

```
s_int32_t
g8032_api_erp_inst_set_ring_id (struct g8032_erps_instance *inst,
                                u_int8_t ring_id)
```

Input Parameters

`inst`	ERP instance
`ring_id`	Ring identifier

Output Parameters

None

Return Values

G8032_ERROR_INVALID_RING_ID when `ring-id` is not in the range <1-254>

G8032_ERROR_NONE when the function succeeds

G8032_ERROR_ERPS_INST_NOT_FOUND when `inst` is NULL

g8032_api_erp_inst_set_rpl_role

This function sets the RPL (Ring Protection Link) role of the ring node.

This function implements the `rpl role` command.

Syntax

```
s_int32_t
g8032_api_erp_inst_set_rpl_role (u_int8_t is_conf,
                                 struct g8032_erps_instance *inst,
                                 enum g8032_instance_role role,
                                 u_int8_t is_east)
```

Input Parameters

`is_conf` Reserved for future use

`inst` ERP instance

`role` Role; one of these constants from the `g8032_instance_role` enum in `onmd/cfm/g8032v2/g8032_erps.h`:

- `G8032_INST_ROLE_NONE` Ring node does not have an RPL role
- `G8032_INST_ROLE_NONOWNER` Ring node does not own the RPL
- `G8032_INST_ROLE_OWNER` Ring node is the RPL owner
- `G8032_INST_ROLE_NEIGHBOUR` Ring node is neighbor to the RPL owner
- `G8032_INST_ROLE_NEXT_NEIGHBOUR` Ring node is neighbor to the neighbor of the RPL owner

`is_east` Whether to assign the role to the east or west interface:

- `PAL_TRUE` Assign role to east interface
- `PAL_FALSE` Assign role to west interface

Output Parameters

None

Return Values

G8032_ERROR_NONE when the function succeeds

G8032_ERROR_ERPS_INST_NOT_FOUND when `inst` is NULL

g8032_api_erp_inst_set_version

This function sets the version to carry in RAPS messages for an interface.

This function implements the `version` command.

Syntax

```
s_int32_t
g8032_api_erp_inst_set_version(u_int8_t is_conf,
                               struct g8032_erps_instance *inst,
                               u_int8_t version,
                               u_int32_t is_east)
```

Input Parameters

`is_conf`	Reserved for future use
`inst`	ERPS instance
`version`	Version
`is_east`	Whether to assign the version to the east or west interface:
`PAL_TRUE`	Assign version to east interface
`PAL_FALSE`	Assign version to west interface

Output Parameters

None

Return Values

G8032_ERROR_NONE when the function succeeds

G8032_ERROR_ERPS_INST_NOT_FOUND when `inst` is NULL

G8032_ERROR_PHY_RING_NOT_ASSOCIATED_WITH_INSTANCE when a physical ring is not associated with the instance

g8032_api_erp_inst_set_virt_channel

This function creates and deletes a virtual channel or a non-virtual channel.

This function implements the `virtual-channel` and `non-virtual-channel` commands.

Syntax

```
s_int32_t
g8032_api_erp_inst_set_virt_channel (u_int8_t is_conf,
                                     struct g8032_erps_instance *inst,
                                     u_int8_t is_virtual,
                                     u_int16_t vid,
                                     u_char *associate_to_inst_name)
```

Input Parameters

`is_conf`	Whether creating or deleting a channel:
`PAL_TRUE`	Creating a channel
`PAL_FALSE`	Deleting a channel
`inst`	ERP instance
`is_virtual`	Whether this action is for a virtual channel or non-virtual channel:
`PAL_TRUE`	Virtual channel
`PAL_FALSE`	Non-virtual channel
`vid`	VLAN identifier.Only specify this parameter when creating a virtual channel; otherwise specify zero.
`associate_to_inst_name`	Adjacent instance name to which the virtual channel is attached. Only specify this parameter when creating a virtual channel; otherwise specify NULL.

Output Parameters

None

Return Values

G8032_ERROR_NONE when the function succeeds

G8032_ERROR_ASSOCIATED_ERPS_INST_NOT_FOUND when `associate_to_inst_name` cannot be found

G8032_ERROR_VIRTUAL_CONF_ALREADY_CONFIGURED when a virtual channel already exists for the instance

G8032_ERROR_NON_VIRTUAL_CONF_NOT_ALLOWED_NOT_SUBRING when the instance is a major ring

G8032_ERROR_NON_VIRTUAL_CONF_NOT_ALLOWED when trying to create a virtual channel but the instance is already configured with a non-virtual channel or when trying to create a non-virtual channel but the instance is already configured with a virtual channel

G8032_ERROR_ERP_INST_VLAN_NOT_CONFIGURED when `vid` is not associated with the bridge

G8032_ERROR_PHY_RING_NOT_ASSOCIATED_WITH_INSTANCE when a physical ring is not associated with the instance

G8032_ERROR_DATA_TRAFFIC_ALREADY_CONFIGURED when the instance is configured for data traffic

G8032_ERROR_RAPS_CHANNEL_ALREADY_CONFIGURED when the `inst` channel is already configured on `associate_to_inst` or the ring-id of `inst` is the same as `associate_to_inst`

G8032_ERROR_ALLOCATION_FAILED when there is an error allocating memory

G8032_ERROR_UNKNOWN when `inst` is NULL

G8032_ERROR_ERPS_INST_NOT_FOUND when `inst` is NULL

g8032_api_erp_inst_show

This function gets details about an ERP instance.

This function implements the `show g8032 erp-instance` command.

Syntax

`s_int32_t`

```
g8032_api_erp_inst_show (u_int8_t is_conf,
                         u_char *br_name,
                         u_char *inst_name,
                         u_char *buf,
                         u_int32_t buf_len)
```

Input Parameters

is_conf	Reserved for future use
br_name	Bridge identifier
inst_name	ERP instance name
buf_len	Buffer length

Output Parameters

buf	Buffer holding ERP instance details

Return Values

G8032_ERROR_NONE when the function succeeds

G8032_ERROR_BRIDGE_NOT_FOUND when the bridge cannot be found

G8032_ERROR_ERPS_INST_NOT_FOUND when the instance cannot be found

g8032_api_phy_ring_create

This function creates or deletes a physical ring.

This function implements the `bridge (<1-32> | backbone) g8032 physical-ring` command.

Syntax

```
s_int32_t
g8032_api_phy_ring_create (u_int8_t is_conf,
                           u_char *br_name,
                           u_char *ring_name,
                           struct interface *east,
                           struct interface *west)
```

Input Parameters

is_conf	Whether creating or deleting a ring:
PAL_TRUE	Creating a ring
PAL_FALSE	Deleting a ring
br_name	Bridge name
ring_name	Name of the ring
east	East interface
west	West interface

Output Parameters

None

Return Values

G8032_ERROR_BRIDGE_NOT_FOUND when the bridge is not found

G8032_ERROR_RAPS_PHY_RING_ALREADY_CONFIGURED when creating a ring and `ring_name` already exists

G8032_ERROR_PHY_RING_NOT_FOUND when deleting a ring and `ring_name` does not exist

G8032_ERROR_PHY_RING_MAX_CONFIG_ALLOWED when the given interfaces are already associated to a physical ring

G8032_ERROR_NONE when the function succeeds

g8032_api_phy_ring_show

This function gets details about a physical ring.

This function implements the `show g8032 physical-ring` command.

Syntax

```
s_int32_t
g8032_api_phy_ring_show (u_int8_t is_conf,
                         u_char *br_name,
                         u_char *ring_name,
                         u_char* buf,
                         u_int32_t buf_len)
```

Input Parameters

`is_conf`	Reserved for future use
`br_name`	Bridge identifier
`ring_name`	Physical ring name
`buf_len`	Buffer length

Output Parameters

`buf`	Buffer holding physical ring details

Return Values

G8032_ERROR_NONE when the function succeeds

G8032_ERROR_BRIDGE_NOT_FOUND when the bridge cannot be found

G8032_ERROR_PHY_RING_NOT_FOUND when the physical ring cannot be found

g8032_api_profile_create

This function creates or deletes a profile.

This function implements the `g8032 profile` command.

Syntax

```
s_int32_t
g8032_api_profile_create (u_int8_t is_conf,
                          u_char *br_name,
                          u_char *profile_name,
```

```
                        struct g8032_instance_profile **profile)
```

Input Parameters

`is_conf`	Whether creating or deleting a profile:
`PAL_TRUE`	Creating a profile
`PAL_FALSE`	Deleting a profile
`br_name`	Bridge name
`profile_name`	Profile name

Output Parameters

`profile`	Profile

Return Values

G8032_ERROR_NONE when the function succeeds or when creating a profile and `profile_name` already exists

G8032_ERROR_PROFILE_CREATE_FAILED when memory allocation fails

G8032_ERROR_BRIDGE_NOT_FOUND when deleting a profile and `br_name` is NULL

G8032_ERROR_PROFILE_NOT_FOUND when deleting a profile and `profile_name` is NULL

g8032_api_profile_set_revertive

This function sets the revertive behavior of the ring node.

This function implements the `enable (revertive | non-revertive)` command.

Syntax

```
s_int32_t
g8032_api_profile_set_revertive (u_int8_t is_conf,
                                 struct g8032_instance_profile *prof,
                                 u_int8_t is_revertive)
```

Input Parameters

`is_conf`	Reserved for future use
`profile`	Profile
`is_revertive`	Whether the ring node is revertive:
`PAL_TRUE`	Ring node is revertive
`PAL_FALSE`	Ring node is non-revertive

Output Parameters

None

Return Values

G8032_ERROR_NONE when the function succeeds

G8032_ERROR_PROFILE_NOT_FOUND when the `profile` is NULL

g8032_api_profile_set_timer

This function sets a timer.

This function implements the `timer` command.

Syntax

```
s_int32_t
g8032_api_profile_set_timer (u_int8_t is_conf,
                             struct g8032_instance_profile *profile,
                             enum g8032_timer_type type,
                             u_int32_t timeout_val)
```

Input Parameters

`is_conf` Reserved for future use

`profile` Profile

`type` Type of timer: one of these constants from the `g8032_timer_type` enum in `onmd/cfm/g8032v2/g8032_profile.h`:

`G8032_TIMER_TYPE_WTR`

Wait-to-restore timer in seconds; used to verify that a signal failure is not intermittent

`G8032_TIMER_TYPE_WTB`

Wait-to-block timer in seconds

`G8032_TIMER_TYPE_GUARD`

Guard timer in microseconds; blocks latent outdated messages from causing unnecessary state changes

`G8032_TIMER_TYPE_HOLDOFF`

Hold-off timer in microseconds; filters intermittent link faults

`timeout_val` Timeout value

Output Parameters

None

Return Values

G8032_ERROR_NONE when the function succeeds

G8032_ERROR_PROFILE_NOT_FOUND when the `profile` is NULL

G8032_ERROR_PROFILE_INVALID_TIMER_TYPE when `type` is not valid

g8032_api_profile_show

This function gets details about a profile.

This function implements the `show g8032 profile` command.

Syntax

```
s_int32_t
g8032_api_profile_show (u_int8_t is_conf,
```

```
                    u_char *br_name,
                    u_char *profile_name,
                    u_char *buf,
                    u_int32_t buf_len)
```

Input Parameters

`is_conf`	Reserved for future use
`br_name`	Bridge identifier
`profile_name`	Profile name
`buf_len`	Buffer length

Output Parameters

`buf`	Buffer holding profile details

Return Values

G8032_ERROR_NONE when the function succeeds

G8032_ERROR_BRIDGE_NOT_FOUND when the bridge cannot be found

G8032_ERROR_PROFILE_NOT_FOUND when the profile cannot be found

g8032_api_tcn_propogation

This function enables or disables TCN (topology change notification) propagation for an interconnected ring.

This function implements the `tcn-propogation` command.

Syntax

```
s_int32_t
g8032_api_tcn_propogation (bool_t conf, struct g8032_erps_instance *inst)
```

Input Parameters

`is_conf`	Whether enabling or disabling TCN propagation:
`PAL_TRUE`	Enable TCN propagation
`PAL_FALSE`	Disable TCN propagation
`inst`	ERPS instance

Output Parameters

None

Return Values

G8032_ERROR_NONE when the function succeeds

G8032_ERROR_ERPS_INST_NOT_FOUND when `inst` is NULL

G8032_ERROR_TCN_FLAG_ALLREADY_SET when trying to enable propagation, but it is already enabled

G8032_ERROR_TCN_FLAG_ALLREADY_CLEAR when trying to disable propagation, but it is already disabled

G8032_ERROR_MAJOR_RING_INSTANCE_NOT_FOUND when a virtual channel not configured

G8032_ERROR_NOT_A_CONNECTING_INSTANCE when there is not an interconnection node for the sub-ring

CHAPTER 10 Ethernet to the First Mile

Ethernet to the First Mile (EFM) is a set of extensions to the IEEE 802.3 MAC (Media Access Control) sublayer. The EFM protocol defines:

- How Ethernet can be transmitted over copper and fiber media used in wide area networks (WAN) and metropolitan area networks (MAN)
- Ethernet Operations, Administration, and Management (OAM)

Ethernet OAM

Ethernet OAM is a point-to-point link-level protocol that monitors the health of a link. It helps determine the location of a failing link or fault condition and complements applications in higher layers. ZebOS-XP EFM provides support for the following features:

- Discovery: This first phase of Ethernet OAM sets the OAM link, where OAM peers exchange their capabilities. The link is established when both peers are satisfied.
- Remote loopback: An OAM entity can put its remote peer into loopback mode. This helps ensure the quality of links during installation or troubleshooting.
- Link Monitoring: This is used to report the link fault conditions of a remote OAM peer under a variety of conditions. It also gives information to the local management applications about link fault conditions.
- Event Logging: When local and remote defects are detected, all the information related to the event is recorded (that is, there is a timestamp of the event, a threshold under which the event is generated, an error count, and so on).
- Retrieving SNMP MIB variables: Described in SNMP API on page 277.

EFM OAM Sublayer

An EFM OAM) sublayer provides mechanisms for monitoring link operations, for example, remote fault detection and loopback controls. It enables network monitors to dynamically track the health of connections and locate failing links or fault conditions. The OAM sublayer resides between a superior sublayer, such as a MAC client, and a subordinate sublayer, for example, the MAC control layer. The OAM sublayer supports interlayer service interfaces. Interfaces are supported between the OAM and the MAC clients to and from the OAM sublayer. In turn, the OAM sublayer communicates with the subordinate layers: MAC Control (optional), MAC, and the physical layer.

- The multiplexer in the OAM sublayer passes a properly formed OAM request to the underlying layer.
- The parser in the OAM sublayer decodes frames received from the subordinate sublayer, passes OAMPDUs to the control function, MAC client frames to the superior sublayer, and loopback fames to the multiplexer function.

OAM Events

A set of events has been defined for EFM OAM that has an impact on link operations. They include

- Critical Link Events. These events are carried within the flag field of each OAMPDU (OAM Protocol Data Unit).
- Link Events. Link events are signaled via Link Event TLVs.
- Local Event Procedure
- Remote Event Procedure

Both local and remote event procedures are communicated via either a critical link event or an event notification OAMPDU that contains a link event TLV.

OAM Protocol Modules

A discovery mechanism helps to detect the presence of an OAM sublayer at the remote client. The OAM sublayer entities implement OAM Discovery. The Local Client that is in active mode initiates the discovery process to the Remote client. If the local client is satisfied with the setting of the Remote client, the Discovery process is completed. If an unsatisfactory response is received from the remote client, the Discovery process does not complete successfully. The Local Stable flag and Local Evaluating flags are used in the Discovery process. When both the local and remote clients are in the Active mode, either of them can send Local and the Remote TLV. A client the in the Passive mode cannot initiate a discovery process.

Remote Loopback

OAM provides a data link layer, frame-level loopback mode, which is controlled remotely. This feature is used for fault localization and link performance testing. Fault localization and link performance testing can be accomplished with the Remote Loopback function. The `local_mux_action` and `local_par_action` parameters are implemented in the software forwarder.

OAM Protocol Data Units

The OAMPDU that are implemented in ZebOS-XP EFM include:

- **Information** — This OAMPDU is identified when the code field is set to 0X00, and is used to send OAM state information to the remote client. If the data field local_pdu of this OAMPDU is set to `lf_info`, it means that no information is passed. If local_pdu is not set to `lf_info`, it means that information is being sent
- **Event Notification** — This OAMPDU is identified when the code field is set to 0X01. It is used to alert remote clients of introduced link events.

  Events are identified when corresponding bits in the flag field are set. The table below provides details.

Table 10-1: Events

Bits	Event Name	Event Description
2	Critical Event	1 - A Critical Event has occurred 0 - A Critical Event has not occurred
1	Dying Gasp	1 - An unrecoverable local failure has occurred 0 - An unrecoverable local failure has not occurred
0	Link Fault	1 - Local Device receive path has detected a fault 0 - Local Device receive path has not detected a fault

- **Loopback Control** — This OAMPDU is identified when the code field is set to 0X04, and is used to control the remote client's OAM remote loopback state. Based on the value in the Remote Loopback Command, which is in the Data field, Loopback OAMPDUs are handled. Refer to the table below for details.

Table 10-2: Loopback OAMPDUs

Command	Description
0x00	Reserved - not transmitted, should be ignored on reception by OAM client.
0x01	Enable OAM Remote Loopback.
0x02	Disable OAM Remote Loopback.
0x03-0xFF	Reserved - not transmitted, should be ignored on reception by OAM client.

- **Organization-Specific** — This OAMPDU is identified when the code field is set to 0XFE. It is used for organization- specific extensions.

OAM Type Length Values

OAM TLVs contain a single octet Type field and single octet Length field. ZebOS-XP EFM OAM implements the following TLVs:

- Information TLV
- Local Information TLV
- Remote Information TLV
- Link Event TLV
- Organization-Specific Event TLV

For in depth details about these OAM TLVs, please refer to section 57.5 of the IEEE 802.1ad - 2004 specification.

EFM OAM API

The functions in this section support EFM OAM (Ethernet to the First Mile Operations, Administration and Management). These functions are called from the:

- Ethernet OAM commands in the *Carrier Ethernet Command Line Interface Reference Guide*
- EFM MIB support functions
- SMI

Function	Description
efm_oam_control_group_attribs	Gets the setting of an attribute in the OAM control group
efm_oam_disable_if_event_set	Specifies an action to take when a remote event is detected
efm_oam_err_event_notify_control_get	Gets an OAM event notification
efm_oam_err_event_notify_control_set	Enables or disables notification when a specified event occurs
efm_oam_err_frame_high_thres_get	Gets the high threshold for the errored frame period event
efm_oam_err_frame_high_thres_set	Sets the high threshold for the errored frame period event

Function	Description
efm_oam_err_frame_low_thres_get	Gets the low threshold for the errored frame event
efm_oam_err_frame_low_thres_set	Sets the low threshold for the errored frame event
efm_oam_err_frame_period_high_thres_set	Sets the high threshold for the errored frame period event
efm_oam_err_frame_period_low_thres_get	Gets the low threshold of the errored frame period event
efm_oam_err_frame_period_low_thres_set	Sets the low threshold of the errored frame period event
efm_oam_err_frame_period_window_get	Gets the window size of the errored frame period event
efm_oam_err_frame_period_window_set	Sets the window size of the errored frame period event
efm_oam_err_frame_second_high_thres_set	Sets the high threshold for the errored frame seconds summary event
efm_oam_err_frame_second_low_thres_get	Gets the low threshold of the errored frame seconds summary event
efm_oam_err_frame_second_low_thres_set	Sets the low threshold of the errored frame seconds summary event
efm_oam_err_frame_second_window_get	Gets the window size of the errored frame seconds summary event
efm_oam_err_frame_second_window_set	Sets the window size of the errored frame seconds summary event.
efm_oam_err_frame_window_get	Gets the window size of the errored frame event
efm_oam_err_frame_window_set	Sets the window size of the errored frame event
efm_oam_get_loopback_ignore_rx	Gets the status of the loopback ignore receive control
efm_oam_get_loopback_status	Gets the loopback status
efm_oam_get_oper_status	Gets the status of the discovery phase between OAM clients on a link
efm_oam_get_stats_group_attribs	Gets the value of the given OAM attribute
efm_oam_link_monitoring_enable_set	Starts or stops the link-monitoring support
efm_oam_link_monitor_support_get	Gets the link monitoring support setting
efm_oam_link_monitor_support_set	Sets the link monitoring support setting
efm_oam_max_rate_get	Gets the maximum number of PDUS that can be transmitted per second
efm_oam_max_rate_set	Sets the maximum number of PDUS that can be transmitted per second
efm_oam_mode_active_set	Sets the OAM mode for an interface
efm_oam_mode_get	Gets the OAM mode for an interface
efm_oam_mode_passive_set	Sets an interface to OAM active mode
efm_oam_pdu_timer_get	Gets the value of the PDU timer
efm_oam_pdu_timer_set	Sets the value of the PDU timer
efm_oam_peer_group_attribs	Gets the value of a peer group numeric attribute

Function	Description
efm_oam_peer_group_strings	Gets the value of a peer group string attribute
efm_oam_protocol_disable	Disables OAM for the specified interface
efm_oam_protocol_enable	Enables OAM for the specified interface
efm_oam_remote_loopback_get	Gets the remote loopback setting
efm_oam_remote_loopback_set	Sets the remote loopback setting
efm_oam_remoteloopback_start	Starts remote loopback for the specified interface
efm_oam_remoteloopback_stop	Stops remote loopback for the specified interface
efm_oam_remote_loopback_timeout_get	Gets the value of the remote loopback timeout.
efm_oam_remote_loopback_timeout_set	Sets the remote loopback timeout interval.
efm_oam_set_link_timer	Sets the minimum number of PDUs that can be transmitted per second by an interface
efm_oam_set_loopback_ignore_rx	Sets the status of the loopback ignore receive control
efm_oam_set_loopback_status	Starts or ends loopback
efm_oam_show_discovery	Gets discovery information for the interface
efm_oam_show_event_log	Gets an interface's event log
efm_oam_show_interface	Gets Ethernet OAM statistics
efm_oam_show_statistics	Gets OAM statistics for the specified interface
efm_oam_show_status	Gets the interface status
efm_oam_sym_period_high_thres_get	Gets the high word of the 8-octet threshold for the errored symbol period event
efm_oam_sym_period_high_thres_set	Sets the high word of the 8-octet threshold for the errored symbol period event
efm_oam_sym_period_high_window_get	Gets the high word of the 8-octet window size for the errored symbol period event
efm_oam_sym_period_high_window_set	Sets the high word of the 8-octet window size for the errored symbol period event
efm_oam_sym_period_low_thres_get	Gets the low word of the 8-octet threshold for the errored symbol period event
efm_oam_sym_period_low_thres_set	Sets the low word of the 8-octet threshold for the errored symbol period event.
efm_oam_sym_period_low_window_get	Gets the low word of the 8-octet window size for the errored symbol period event

Function	Description
efm_oam_sym_period_low_window_set	Gets the low word of the 8-octet window size for the errored symbol period event.
efm_oam_sym_period_window_set	Sets the size of the window for the errored symbol period event

Include File

Except where noted otherwise, you need to include `onmd/efm-oam/efm_api.h` to call the functions in this chapter.

efm_oam_control_group_attribs

This function gets the setting of an attribute in the OAM control group.

Syntax

```
s_int32_t
efm_oam_control_group_attribs (struct interface *ifp, u_int32_t *attrib,
                               u_int8_t attrib_type)
```

Input Parameters

`ifp` Pointer to the interface structure

`attrib_type` The control group variable; one of these constants from onmd/efm-oam/efm_snmp.h:

`DOT3OAMMAXOAMPDUSIZE`

Maximum OAM PDU size

`DOT3OAMCONFIGREVISION`

Configuration revision

`DOT3OAMFUNCTIONSSUPPORTED`

Functions supported in a SNMP BITS data type:

`unidirectionalSupport(0)`

OAM entity can send OAMPDUs on links that are operating in unidirectional mode

`loopbackSupport(1)`

OAM entity can send and receive OAM loopback commands

`eventSupport(2)`

OAM entity can send and receive event OAMPDUs to signal various error conditions

`variableSupport(3)`

OAM entity can send and receive variable request and response OAMPDUs

Output Parameters

`attrib` The setting of the variable

Return Values

EFM_FAILURE when the interface is not found or the `attrib_type` is invalid

EFM_OAM_ERR_IF_NOT_FOUND when the onmd interface is not found

EFM_OAM_ERR_NOT_ENABLED when EFM is not enabled for the interface

EFM_OAM_ERR_INTERNAL when there is an internal error

EFM_SUCCESS when the function succeeds

efm_oam_disable_if_event_set

This function specifies an action to take when a remote event is detected.

This function implements the following commands:

```
ethernet oam link-monitor high threshold action error-disable-interface
ethernet oam remote-failure critical-event|dying-gasp|link-fault action error-
  disable-interface
```

Syntax

```
s_int32_t
efm_oam_disable_if_event_set (struct interface *ifp, u_int8_t event, u_int8_t enable)
```

Input Parameters

`ifp`	Pointer to the interface structure
`event`	Remote event; one of these constants from `onmd/efm-oam/efm_oam.h`:
	`EFM_DISABLE_IF_ON_REM_CRITICAL_EVENT`
	`EFM_DISABLE_IF_ON_REM_DYING_GASP`
	`EFM_DISABLE_IF_ON_REM_LINK_FAULT`
	`EFM_DISABLE_IF_ON_HIGH_THRESHOLD`
`enable`	Specify PAL_TRUE to disable the interface when the event occurs; specify PAL_FALSE to reverse this behavior and *not* disable the interface when the event occurs

Output Parameters

None

Return Values

EFM_OAM_ERR_IF_NOT_FOUND when the onmd interface is not found

EFM_OAM_ERR_NOT_ENABLED when EFM is not enabled for the interface

EFM_OAM_ERR_INTERNAL when there is an internal error

EFM_SUCCESS when the function succeeds

efm_oam_err_event_notify_control_get

This function gets an OAM event notification.

The EFM MIB support calls this function.

Syntax

```
s_int32_t
```

```
efm_oam_err_event_notify_control_get (struct interface *ifp, u_int32_t *attrib,
                                      u_int8_t attrib_type)
```

Input Parameters

`ifp` Pointer to the interface structure

`attrib_type` Attribute identifier; one of these constants from onmd/efm-oam/efm_snmp.h:

`DOT3OAMERRSYMPERIODEVNOTIFENABLE`

Send an Event Notification PDU when an Errored Symbol Period Event occurs

`DOT3OAMERRFRAMEPERIODEVNOTIFENABLE`

Send an Event Notification PDU when an Errored Frame Period Event occurs

`DOT3OAMERRFRAMEEVNOTIFENABLE`

Send an Event Notification PDU when an Errored Frame Event occurs

`DOT3OAMERRFRAMESECSEVNOTIFENABLE`

Send an Event Notification PDU when an Errored Frame Seconds Event occurs

`DOT3OAMDYINGGASPENABLE`

Indicate a dying gasp via the PDU flags field to the peer OAM entity

`DOT3OAMCRITICALEVENTENABLE`

Indicate a critical event via the PDU flags to the peer OAM entity

Output Parameters

`attrib` Value of the attribute; one of these constants from the enum ev_notify_enable in onmd/efm-oam/efm_oam.h:

`EFM_OAM_NOTIFY_ON`

Event notification is on

`EFM_OAM_NOTIFY_OFF`

Event notification if off

Return Values

EFM_FAILURE when the interface is null or the attrib_type is invalid

EFM_OAM_ERR_IF_NOT_FOUND when the onmd interface is not found

EFM_OAM_ERR_NOT_ENABLED when EFM is not enabled for the interface

EFM_OAM_ERR_INTERNAL when there is an internal error

EFM_SUCCESS when the function succeeds

efm_oam_err_event_notify_control_set

This function enables or disables notification when a specified event occurs.

The EFM MIB support calls this function.

Syntax

```
s_int32_t
efm_oam_err_event_notify_control_set (struct interface *ifp, u_int32_t attrib,
```

```
                                    u_int8_t attrib_type)
```

Input Parameters

`ifp` Pointer to the interface structure

`attrib_type` Attribute identifier; one of these constants from onmd/efm-oam/efm_snmp.h:

`DOT3OAMERRSYMPERIODEVNOTIFENABLE`

Send an Event Notification PDU when an Errored Symbol Period Event occurs

`DOT3OAMERRFRAMEPERIODEVNOTIFENABLE`

Send an Event Notification PDU when an Errored Frame Period Event occurs

`DOT3OAMERRFRAMEEVNOTIFENABLE`

Send an Event Notification PDU when an Errored Frame Event occurs

`DOT3OAMERRFRAMESECSEVNOTIFENABLE`

Send an Event Notification PDU when an Errored Frame Seconds Event occurs

`DOT3OAMDYINGGASPENABLE`

Indicate a dying gasp to the peer OAM entity via the PDU flags field

`DOT3OAMCRITICALEVENTENABLE`

Indicate a critical event to the peer OAM entity via the PDU flags field

`attrib` Value of the attribute; one of these constants from the enum ev_notify_enable in onmd/efm-oam/efm_oam.h:

`EFM_OAM_NOTIFY_ON`

Turn on event notification

`EFM_OAM_NOTIFY_OFF`

Turn off event notification

Output Parameters

None

Return Values

EFM_FAILURE when the interface is null or the attrib_type is invalid

EFM_OAM_ERR_IF_NOT_FOUND when the onmd interface is not found

EFM_OAM_ERR_NOT_ENABLED when EFM is not enabled for the interface

EFM_OAM_ERR_INTERNAL when there is an internal error

EFM_SUCCESS when the function succeeds

efm_oam_err_frame_high_thres_get

This function gets the high threshold for the errored frame period event.

The SMI calls this function.

Syntax

```
s_int32_t
efm_oam_err_frame_high_thres_get (struct interface *ifp, int *thres_high)
```

Input Parameters

`ifp`	Pointer to the interface structure

Output Parameters

`thres_high`	The high threshold for the errored frame period event

Return Values

EFM_OAM_ERR_IF_NOT_FOUND when the onmd interface is not found

EFM_OAM_ERR_NOT_ENABLED when EFM is not enabled for the interface

EFM_OAM_ERR_INTERNAL when there is an internal error

EFM_SUCCESS when the function succeeds

efm_oam_err_frame_high_thres_set

This function sets the high threshold for the errored frame period event.

This function implements the `ethernet oam link-monitor threshold frame high THRES_HIGH` command. The SMI also calls this function.

Syntax

```
s_int32_t
efm_oam_err_frame_high_thres_set (struct interface *ifp, u_int16_t thres_high)
```

Input Parameters

`ifp`	Pointer to the interface structure
`thres_high`	The high threshold for the errored frame period event

Output Parameters

None

Return Values

EFM_OAM_ERR_IF_NOT_FOUND when the onmd interface is not found

EFM_OAM_ERR_NOT_ENABLED when EFM is not enabled for the interface

EFM_OAM_ERR_INTERNAL when there is an internal error

EFM_SUCCESS when the function succeeds

efm_oam_err_frame_low_thres_get

This function gets the low threshold for the errored frame event.

The EFM MIB support calls this function.

Syntax

```
s_int32_t
efm_oam_err_frame_low_thres_get (struct interface *ifp, int *thres_low)
```

Input Parameters

`ifp`	Pointer to the interface structure

Output Parameters

`thres_low`	The low threshold for the errored frame event

Return Values

EFM_OAM_ERR_IF_NOT_FOUND when the onmd interface is not found

EFM_OAM_ERR_NOT_ENABLED when EFM is not enabled for the interface

EFM_OAM_ERR_INTERNAL when there is an internal error

EFM_SUCCESS when the function succeeds

efm_oam_err_frame_low_thres_set

This function sets the low threshold for the errored frame event.

This function implements the `ethernet oam link-monitor frame threshold low THRES_LOW` value command. The EFM MIB support also calls this function.

Syntax

```
s_int32_t
efm_oam_err_frame_low_thres_set (struct interface *ifp, u_int16_t thres_low)
```

Input Parameters

`ifp`	Pointer to the interface structure
`thres_low`	The low threshold for the errored frame event

Output Parameters

None

Return Values

EFM_OAM_ERR_IF_NOT_FOUND when the onmd interface is not found

EFM_OAM_ERR_NOT_ENABLED when EFM is not enabled for the interface

EFM_OAM_ERR_INTERNAL when there is an internal error

EFM_SUCCESS when the function succeeds

efm_oam_err_frame_period_high_thres_set

This function sets the high threshold for the errored frame period event.

This function implements the `ethernet oam link-monitor threshold frame-period high THRES_HIGH` command.

Syntax

```
s_int32_t
efm_oam_err_frame_period_high_thres_set (struct interface *ifp, u_int16_t thres_high)
```

Input Parameters

`ifp`	Pointer to the interface structure
`thres_high`	The high threshold for the errored frame period event

Output Parameters

None

Return Values

EFM_OAM_ERR_IF_NOT_FOUND when the onmd interface is not found

EFM_OAM_ERR_NOT_ENABLED when EFM is not enabled for the interface

EFM_OAM_ERR_INTERNAL when there is an internal error

EFM_SUCCESS when the function succeeds

efm_oam_err_frame_period_low_thres_get

This function gets the low threshold of the errored frame period event.

The EFM MIB support calls this function.

Syntax

```
s_int32_t
efm_oam_err_frame_period_low_thres_get (struct interface *ifp, u_int32_t *thres_low)
```

Input Parameters

`ifp`	Pointer to the interface structure

Output Parameters

`thres_low`	The low threshold of the errored frame period event

Return Values

EFM_OAM_ERR_IF_NOT_FOUND when the onmd interface is not found

EFM_OAM_ERR_NOT_ENABLED when EFM is not enabled for the interface

EFM_OAM_ERR_INTERNAL when there is an internal error

EFM_SUCCESS when the function succeeds

efm_oam_err_frame_period_low_thres_set

This function sets the low threshold of the errored frame period event.

This function implements the `ethernet oam link-monitor frame-period threshold low THRES_LOW` value command. The EFM MIB support also calls this function.

Syntax

```
s_int32_t
efm_oam_err_frame_period_low_thres_set (struct interface *ifp, u_int32_t thres_low)
```

Input Parameters

`ifp`	Pointer to the interface structure
`thres_low`	The low threshold of the errored frame period event

Output Parameters

None

Return Values

EFM_OAM_ERR_IF_NOT_FOUND when the onmd interface is not found

EFM_OAM_ERR_NOT_ENABLED when EFM is not enabled for the interface

EFM_OAM_ERR_INTERNAL when there is an internal error

EFM_SUCCESS when the function succeeds

efm_oam_err_frame_period_window_get

This function gets the window size of the errored frame period event.

The EFM MIB support calls this function.

Syntax

```
s_int32_t
efm_oam_err_frame_period_window_get (struct interface *ifp, int *window)
```

Input Parameters

`ifp`	Pointer to the interface structure

Output Parameters

`window`	The window size of the errored frame period event

Return Values

EFM_FAILURE when the interface is NULL.

EFM_OAM_ERR_IF_NOT_FOUND when the onmd interface is not found

EFM_OAM_ERR_NOT_ENABLED when EFM is not enabled for the interface

EFM_OAM_ERR_INTERNAL when there is an internal error

EFM_SUCCESS when the function succeeds

efm_oam_err_frame_period_window_set

This function sets window size of the errored frame period event.

This function implements the `ethernet oam link-monitor frame-period window WINDOW` command. The EFM MIB support also calls this function.

Syntax

```
s_int32_t
efm_oam_err_frame_period_window_set (struct interface *ifp, u_int16_t window)
```

Input Parameters

`ifp`	Pointer to the interface structure
`window`	The size of the window for the errored frame period event

Output Parameters

None

Return Values

EFM_OAM_ERR_IF_NOT_FOUND when the onmd interface is not found

EFM_OAM_ERR_NOT_ENABLED when EFM is not enabled for the interface

EFM_OAM_ERR_INTERNAL when there is an internal error

EFM_SUCCESS when the function succeeds

efm_oam_err_frame_second_high_thres_set

This function sets the high threshold for the errored frame seconds summary event.

This function implements the `ethernet oam link-monitor threshold frame-seconds high THRES_HIGH` command.

Syntax

```
s_int32_t
efm_oam_err_frame_second_high_thres_set (struct interface *ifp, u_int16_t thres_high)
```

Input Parameters

`ifp`	Pointer to the interface structure
`thres_high`	The high threshold for the errored frame seconds summary event

Output Parameters

None

Return Values

EFM_OAM_ERR_IF_NOT_FOUND when the onmd interface is not found

EFM_OAM_ERR_NOT_ENABLED when EFM is not enabled for the interface

EFM_OAM_ERR_INTERNAL when there is an internal error

EFM_SUCCESS when the function succeeds

efm_oam_err_frame_second_low_thres_get

This function gets the low threshold of the errored frame seconds summary event.

The EFM MIB support calls this function.

Syntax

```
s_int32_t
efm_oam_err_frame_second_low_thres_get (struct interface *ifp, u_int16_t *thres_low)
```

Input Parameters

`ifp`	Pointer to the interface structure

Output Parameters

`thres_low`	The low threshold of the errored frame seconds summary event

Return Values

EFM_OAM_ERR_IF_NOT_FOUND when the onmd interface is not found

EFM_OAM_ERR_NOT_ENABLED when EFM is not enabled for the interface

EFM_OAM_ERR_INTERNAL when there is an internal error

EFM_SUCCESS when the function succeeds

efm_oam_err_frame_second_low_thres_set

This function sets the low threshold of the errored frame seconds summary event.

This function is called by the `ethernet oam link-monitor frame-seconds threshold low THRES_LOW` value command. The EFM MIB support also calls this function.

Syntax

```
s_int32_t
efm_oam_err_frame_second_low_thres_set (struct interface *ifp, u_int16_t thres_low)
```

Input Parameters

`ifp`	Pointer to the interface structure
`thres_low`	The low threshold of the errored frame seconds summary event

Output Parameters

None

Return Values

EFM_OAM_ERR_IF_NOT_FOUND when the onmd interface is not found

EFM_OAM_ERR_NOT_ENABLED when EFM is not enabled for the interface

EFM_OAM_ERR_INTERNAL when there is an internal error

EFM_SUCCESS when the function succeeds

efm_oam_err_frame_second_window_get

This function gets the window size of the errored frame seconds summary event.

The EFM MIB support calls this function.

Syntax

```
s_int32_t
efm_oam_err_frame_second_window_get(struct interface *ifp, u_int32_t* window)
```

Input Parameters

`ifp`	Pointer to the interface structure

Output Parameters

`window`	The size of the window for the errored frame seconds summary event

Return

EFM_OAM_ERR_IF_NOT_FOUND when the onmd interface is not found

EFM_OAM_ERR_NOT_ENABLED when EFM is not enabled for the interface

EFM_OAM_ERR_INTERNAL when there is an internal error

EFM_SUCCESS when the function succeeds

efm_oam_err_frame_second_window_set

This function sets the window size of the errored frame seconds summary event.

This function implements the `ethernet oam link-monitor frame-seconds window WINDOW` command. The EFM MIB support also calls this function.

Syntax

```
s_int32_t
efm_oam_err_frame_second_window_set (struct interface *ifp, u_int16_t window)
```

Input Parameters

`ifp`	Pointer to the interface structure
`window`	The size of the window for the errored frame seconds summary event.

Output Parameters

None

Return Values

EFM_OAM_ERR_IF_NOT_FOUND when the onmd interface is not found

EFM_OAM_ERR_NOT_ENABLED when EFM is not enabled for the interface

EFM_OAM_ERR_INTERNAL when there is an internal error

EFM_SUCCESS when the function succeeds

efm_oam_err_frame_window_get

This function gets the window size of the errored frame event.

The EFM MIB support calls this function.

Syntax

```
s_int32_t
efm_oam_err_frame_window_get(struct interface *ifp, u_int32_t* window)
```

Input Parameters

`ifp`	Pointer to the interface structure

Output Parameters

`window`	The size of the window for the errored frame event

Return

EFM_OAM_ERR_IF_NOT_FOUND when the onmd interface is not found

EFM_OAM_ERR_NOT_ENABLED when EFM is not enabled for the interface

EFM_OAM_ERR_INTERNAL when there is an internal error

EFM_SUCCESS when the function succeeds

efm_oam_err_frame_window_set

This function sets the window size of the errored frame event.

This function implements the `ethernet oam link-monitor frame window WINDOW` command. The EFM MIB support also calls this function.

Syntax

```
s_int32_t
efm_oam_err_frame_window_set (struct interface *ifp, u_int16_t window)
```

Input Parameters

`ifp`	Pointer to the interface structure
`window`	The size of the window for the errored frame event

Output Parameters

None

Return Values

EFM_OAM_ERR_IF_NOT_FOUND when the onmd interface is not found

EFM_OAM_ERR_NOT_ENABLED when EFM is not enabled for the interface

EFM_OAM_ERR_INTERNAL when there is an internal error

EFM_SUCCESS when the function succeeds

efm_oam_get_loopback_ignore_rx

This function gets the status of the loopback ignore receive control of the local OAM client.

The EFM MIB support calls this function.

PI Call

```
s_int32_t
efm_oam_get_loopback_ignore_rx (struct interface *ifp, u_int8_t* ignoreRxStatus)
```

Input Parameters

`ifp` Pointer to the interface structure

Output Parameters

`ignoreRxStatus`

Ignore receive status; one of these constants from the enum lb_control in onmd/efm-oam/efm_oam.h:

`EFM_OAM_LB_REQUEST_IGNORE`

Ignore loopback requests

`EFM_OAM_LB_REQUEST_ACCEPT`

Accept loopback requests

Return

EFM_FAILURE when the interface is null

EFM_OAM_ERR_IF_NOT_FOUND when the onmd interface is not found

EFM_OAM_ERR_NOT_ENABLED when EFM is not enabled for the interface

EFM_OAM_ERR_INTERNAL when there is an internal error

EFM_SUCCESS when the function succeeds

efm_oam_get_loopback_status

This function gets the loopback status of a local OAM client. The loopback status is derived based on the multiplexer and parser action of the local and remote OAM entities, as shown in the table below.

Loopback status	Local Parser Action	Local Multiplexer Action	Remote Parser Action	Remote Multiplexer Action
No loopback	Forward	Forward	Forward	Forward
Init loopback	Discard	Discard	Forward	Forward
Remote loopback	Discard	Forward	Loopback	Discard
Terminating loopback	Discard	Discard	Loopback	Discard
Local loopback	Loopback	Discard	Discard	Forward

The EFM MIB support calls this function.

Syntax

```
s_int32_t
efm_oam_get_loopback_status(struct interface *ifp, u_int8_t* lbStatus)
```

Input Parameters

`ifp` Pointer to the interface structure

Output Parameters

`lbStatus` Loopback status; one of these constants from the efm_lb_status enum in onmd/efm-oam/efm_oam.h:

`EFM_OAM_UNKNOWN`
This is set when this function returns a value other than EFM_SUCCESS

`EFM_OAM_NO_LB`
No loopback

`EFM_OAM_INIT_LB`
Initializing loopback

`EFM_OAM_REMOTE_LB`
In remote loopback

`EFM_OAM_TERMINATING_LB`
Terminating loopback

`EFM_OAM_LOCAL_LBK`
Local DTE is in loopback

Return

EFM_FAILURE when the interface is null

EFM_OAM_ERR_IF_NOT_FOUND when the onmd interface is not found

EFM_OAM_ERR_NOT_ENABLED when EFM is not enabled for the interface

EFM_OAM_ERR_INTERNAL when there is an internal error

EFM_SUCCESS when the function succeeds

efm_oam_get_oper_status

This function is gets the status of the discovery phase between OAM clients on a link. During initialization or failure, two OAM entities on the same link begin a discovery phase to determine what OAM capabilities they can use.

The EFM MIB support calls this function.

Syntax

```
s_int32_t
efm_oam_get_oper_status (struct interface *ifp, u_int32_t *oper_status)
```

Input Parameters

`ifp` Pointer to the interface structure

Output Parameters

`oper_status` The result of discovery; one of these constants from the enum efm_oam_oper_status in onmd/efm-oam/efm_types.h:

`OPER_ACTIVE_SEND_LOCAL`
In active mode and actively trying to discover whether the peer has OAM capability but has not yet made that determination

`OPER_DISABLED`

OAM is disabled on this interface

`OPER_LINK_FAULT`

The link has detected a fault and is transmitting OAM PDUs with a link fault indication or the interface is not operating

`OPER_NONOPER_HALF_DUPLEX`

The interface is in half-duplex operation (OAM functions are not designed to work completely over half-duplex interfaces)

`OPER_OPERATIONAL`

The local and remote OAM entities have accepted the peering

`OPER_PASSIVE_WAIT`

In passive mode and waiting to see if the peer device is OAM capable

`OPER_PEERING_LOCALLY_REJECTED`

Local OAM entity rejected the peering

`OPER_PEERING_REMOTELY_REJECTED`

Remote OAM entity rejected the peering

`OPER_SEND_LOCAL_REMOTE`

Local OAM entity has discovered the peer but has not yet accepted or rejected the configuration of the peer

`OPER_SEND_LOCAL_REMOTE_OK`

Local OAM entity accepted the peering

Return

EFM_SUCCESS when the function succeeds

EFM_OAM_ERR_IF_NOT_FOUND when the onmd interface is not found

efm_oam_get_stats_group_attribs

This function gets the value of the given OAM attribute.

The EFM MIB support calls this function.

Syntax

```
s_int32_t
efm_oam_get_stats_group_attribs (struct interface *ifp, u_int32_t *attrib,
                                 u_int8_t attrib_type
```

Input Parameters

`ifp` Pointer to the interface structure

`attrib_type` Attribute identifier; one of these constants from onmd/efm-oam/efm_snmp.h:

`DOT3OAMINFORMATIONTX`

Information OAM PDUs transmitted on this interface

`DOT3OAMINFORMATIONRX`

Information OAM PDUs received on this interface

DOT3OAMUNIQUEEVENTNOTIFICATIONTX

Unique event OAM PDUs transmitted on this interface

DOT3OAMUNIQUEEVENTNOTIFICATIONRX

Unique event OAM PDUs received on this interface

DOT3OAMLOOPBACKCONTROLTX

Loopback control OAM PDUs transmitted on this interface

DOT3OAMLOOPBACKCONTROLRX

Loopback control OAM PDUs received on this interface

DOT3OAMUNSUPPORTEDCODESTX

OAM PDUs transmitted on this interface with an unsupported operation code

DOT3OAMUNSUPPORTEDCODESRX

OAM PDUs received on this interface with an unsupported operation code

DOT3OAMDUPLICATEEVENTNOTIFICATIONTX

Duplicate event OAM PDUs transmitted

DOT3OAMDUPLICATEEVENTNOTIFICATIONRX

Duplicate event OAM PDUs received

DOT3OAMVARIABLEREQUESTTX

Variable request OAM PDUs transmitted

DOT3OAMVARIABLEREQUESTRX

Variable request OAM PDUs received

DOT3OAMVARIABLERESPONSETX

Variable response OAM PDUs transmitted

DOT3OAMVARIABLERESPONSERX

Variable response OAM PDUs received

DOT3OAMORGSPECIFICTX

Organization specific OAM PDUs transmitted

DOT3OAMORGSPECIFICRX

Organization specific OAM PDUs received

DOT3OAMFRAMESLOSTDUETOOAM

Frames dropped by the multiplexer

Output Parameters

attrib	Value of the attribute; if an attribute is not supported on a platform, then this function sets this parameter to zero

Return

EFM_FAILURE when the interface is null

EFM_OAM_ERR_IF_NOT_FOUND when the onmd interface is not found

EFM_OAM_ERR_NOT_ENABLED when EFM is not enabled for the interface

EFM_OAM_ERR_INTERNAL when there is an internal error

EFM_SUCCESS when the function succeeds

efm_oam_link_monitoring_enable_set

This function starts or stops the link-monitoring support.

This function implements the `ethernet oam link-monitor on` command.

Syntax

```
s_int32_t
efm_oam_link_monitoring_enable_set (struct interface *ifp, u_int8_t enable)
```

Input Parameters

`ifp`	Pointer to the interface structure
`enable`	Whether to start or stop link-monitoring on the interface

Output Parameters

None

Return Values

EFM_SUCCESS when the function succeeds

EFM_FAILURE when the interface is NULL

EFM_OAM_ERR_IF_NOT_FOUND when the onmd interface is not found

EFM_OAM_ERR_INTERNAL when there is an internal error

EFM_OAM_ERR_NOT_ENABLED when EFM is not enabled for the interface

efm_oam_link_monitor_support_get

This function gets the link monitoring support setting.

SMI calls this function.

Syntax

```
s_int32_t
efm_oam_link_monitor_support_get (struct interface *ifp, u_int8_t *enable)
```

Input Parameters

`ifp`	Pointer to the interface structure

Output Parameters

`enable`	Whether link-monitoring support is enabled (PAL_TRUE) or disabled (PAL_FALSE)

Return Values

EFM_OAM_ERR_IF_NOT_FOUND when the onmd interface is not found

EFM_OAM_ERR_NOT_ENABLED when EFM is not enabled for the interface

EFM_OAM_ERR_INTERNAL when there is an internal error

EFM_SUCCESS when the function succeeds

efm_oam_link_monitor_support_set

This function enables or disables link-monitoring support for an interface.

This function implements the `ethernet oam link-monitor supported` command.

SMI calls this function.

Syntax

```
s_int32_t
efm_oam_link_monitor_support_set (struct interface *ifp, u_int8_t enable)
```

Input Parameters

`ifp`	Pointer to the interface structure
`enable`	Whether link-monitoring support is enabled (PAL_TRUE) or disabled (PAL_FALSE)

Output Parameters

None

Return Values

EFM_OAM_ERR_IF_NOT_FOUND when the onmd interface is not found

EFM_OAM_ERR_NOT_ENABLED when EFM is not enabled for the interface

EFM_OAM_ERR_INTERNAL when there is an internal error

EFM_SUCCESS when the function succeeds

efm_oam_max_rate_get

This function gets the maximum number of PDUS that can be transmitted per second.

SMI calls this function.

Syntax

```
s_int32_t
efm_oam_max_rate_get (struct interface *ifp, u_int8_t *max_pdus)
```

Input Parameters

`ifp`	Pointer to the interface structure

Output Parameters

`max_pdus`	The maximum number of PDUS that can be transmitted per second

Return Values

EFM_OAM_ERR_IF_NOT_FOUND when the onmd interface is not found

EFM_OAM_ERR_NOT_ENABLED when EFM is not enabled for the interface

EFM_OAM_ERR_INTERNAL when there is an internal error

EFM_SUCCESS when the function succeeds

efm_oam_max_rate_set

This function sets the maximum number of PDUs that can be transmitted per second by an interface.

This function implements the `ethernet oam max-rate PDUS` command. The SMI also calls this function.

Syntax

```
s_int32_t
efm_oam_max_rate_set (struct interface *ifp, u_int8_t max_pdus)
```

Input Parameters

`ifp`	Pointer to the interface structure
`max_pdus`	The maximum number of PDUS that can be transmitted per second

Output Parameters

None

Return Values

EFM_OAM_ERR_IF_NOT_FOUND when the onmd interface is not found

EFM_OAM_ERR_NOT_ENABLED when EFM is not enabled for the interface

EFM_OAM_ERR_INTERNAL when there is an internal error

EFM_SUCCESS when the function succeeds

efm_oam_mode_active_set

This function sets the OAM mode for an interface.

OAM entities on a link can be in active mode or passive mode:

- An entity in passive mode generally waits for the peer to initiate OAM actions with it
- An entity in active mode initiates monitoring activities with the remote OAM peer entity

This function implements the `ethernet oam mode active` command. The EFM MIB support also calls this function.

Syntax

```
s_int32_t
efm_oam_mode_active_set (struct interface *ifp)
```

Input Parameters

`ifp`	Pointer to the interface structure

Output Parameters

None

Return Values

EFM_OAM_ERR_IF_NOT_FOUND when the onmd interface is not found

EFM_OAM_ERR_NOT_ENABLED when EFM is not enabled for the interface

EFM_OAM_ERR_ALREADY_ACTIVE when the mode is already active

EFM_OAM_ERR_INTERNAL when there is an internal error

EFM_SUCCESS when the function succeeds

efm_oam_mode_get

This function gets the OAM mode for an interface.

The EFM MIB support calls this function.

Syntax

```
s_int32_t
efm_oam_mode_get (struct interface *ifp, int *mode)
```

Input Parameters

`ifp` Pointer to the interface structure

Output Parameters

`mode` OAM mode of the interface; one of these constants from the enum `efm_oam_mode` in `onmd/efm-oam/efm_types.h`:

`EFM_OAM_MODE_ACTIVE`

Entity initiates monitoring activities with the remote OAM peer entity

`EFM_OAM_MODE_PASSIVE`

Entity waits for the peer to initiate OAM actions with it

Return

EFM_FAILURE when the interface is null

EFM_OAM_ERR_IF_NOT_FOUND when the onmd interface is not found

EFM_OAM_ERR_NOT_ENABLED when EFM is not enabled for the interface

EFM_OAM_ERR_INTERNAL when there is an internal error

EFM_SUCCESS when the function succeeds

efm_oam_mode_passive_set

This function sets an interface to OAM active mode.

OAM entities on a link can be in active mode or passive mode:

- An entity in passive mode generally waits for the peer to initiate OAM actions with it
- An entity in active mode initiates monitoring activities with the remote OAM peer entity

This function implements the `ethernet oam mode passive` command. The EFM MIB support also calls this function.

Syntax

```
s_int32_t
efm_oam_mode_passive_set (struct interface *ifp)
```

Input Parameters

`ifp`	Pointer to the interface structure

Output Parameters

None

Return Values

EFM_OAM_ERR_IF_NOT_FOUND when the onmd interface is not found

EFM_OAM_ERR_NOT_ENABLED when EFM is not enabled for the interface

EFM_OAM_ERR_ALREADY_PASSIVE when the mode is already passive

EFM_OAM_ERR_INTERNAL when there is an internal error

EFM_SUCCESS when the function succeeds

efm_oam_pdu_timer_get

This function gets the value of the PDU timer.

SMI calls this function.

Syntax

```
s_int32_t
efm_oam_pdu_timer_get (struct interface *ifp, u_int8_t *secs)
```

Input Parameters

`ifp`	Pointer to the interface structure

Output Parameters

`secs`	Timer value in seconds

Return Values

EFM_OAM_ERR_IF_NOT_FOUND when the onmd interface is not found

EFM_OAM_ERR_NOT_ENABLED when EFM is not enabled for the interface

EFM_OAM_ERR_INTERNAL when there is an internal error

EFM_SUCCESS when the function succeeds

efm_oam_pdu_timer_set

This function sets an interface's EFM timer to a specified number of seconds.

This function implements the `ethernet oam pdu-timer TIMER-VALUE` value command. The SMI also calls this function.

Syntax

```
s_int32_t
efm_oam_pdu_timer_set (struct interface *ifp, u_int8_t secs)
```

Input Parameters

`ifp`	Pointer to the interface structure
`secs`	Timer value in seconds

Output Parameters

None

Return Values

EFM_OAM_ERR_IF_NOT_FOUND when the onmd interface is not found

EFM_OAM_ERR_NOT_ENABLED when EFM is not enabled for the interface

EFM_OAM_ERR_INTERNAL when there is an internal error

EFM_SUCCESS when the function succeeds

efm_oam_peer_group_attribs

This function gets the value of a peer group numeric attribute.

The EFM MIB support calls this function.

Syntax

```
s_int32_t
efm_oam_peer_group_attribs (struct interface *ifp, u_int32_t *attrib,
                            u_int8_t attrib_type)
```

Input Parameters

`ifp` Pointer to the interface structure

`attrib_type` Attribute identifier; one of these constants from onmd/efm-oam/efm_snmp.h:

`DOT3OAMPEERVENDORINFO`
Peer vendor information

`DOT3OAMPEERMODE`
Peer mode (active or passive)

`DOT3OAMPEERMAXOAMPDUSIZE`
Peer maximum OAM PDU size

`DOT3OAMPEERCONFIGREVISION`
Peer configuration version

`DOT3OAMPEERFUNCTIONSSUPPORTED`
Functions supported in a SNMP BITS data type:

`unidirectionalSupport(0)`

Peer OAM entity can send OAM frames when the receive path is inoperable

`loopbackSupport(1)`

Peer OAM entity can send and receive OAM loopback commands

`eventSupport(2)`

Peer OAM entity can send and receive event OAMPDUs to signal various error conditions

`variableSupport(3)`

Peer OAM entity can send and receive variable requests OAMPDUs

Output Parameters

`attrib` Value of the attribute

Return

EFM_FAILURE when the interface is null or the attrib_type is invalid

EFM_OAM_ERR_IF_NOT_FOUND when the onmd interface is not found

EFM_OAM_ERR_NOT_ENABLED when EFM is not enabled for the interface

EFM_OAM_ERR_INTERNAL when there is an internal error

EFM_SUCCESS when the function succeeds

efm_oam_peer_group_strings

This function gets the value of a peer group string attribute.

The EFM MIB support calls this function.

Syntax

```
s_int32_t
efm_oam_peer_group_strings (struct interface *ifp, u_char *string,
                            u_int8_t attrib_type)
```

Input Parameters

`ifp` Pointer to the interface structure

`attrib_type` Attribute identifier; one of these constants from onmd/efm-oam/efm_snmp.h:

`DOT3OAMPEERMACADDRESS`

Peer MAC address

`DOT3OAMPEERVENDOROUI`

Peer vendor organizational unique identifier

Output Parameters

`string` Value of the attribute

Return

EFM_FAILURE when the interface is null or the attrib_type is invalid

EFM_OAM_ERR_IF_NOT_FOUND when the onmd interface is not found

EFM_OAM_ERR_NOT_ENABLED when EFM is not enabled for the interface

EFM_OAM_ERR_INTERNAL when there is an internal error

EFM_SUCCESS when the function succeeds

efm_oam_protocol_disable

This function disables OAM for the specified interface.

This function implements the `ethernet oam disable` command. The EFM MIB support also calls this function.

Syntax

```
s_int32_t
efm_oam_protocol_disable (struct interface *ifp)
```

Input Parameters

`ifp`	Pointer to the interface structure

Output Parameters

None

Return Values

EFM_OAM_ERR_IF_NOT_FOUND when the onmd interface is not found

EFM_OAM_ERR_ALREADY_DISABLED when OAM is already disabled

EFM_OAM_ERR_INTERNAL when there is an internal error

EFM_SUCCESS when the function succeeds

efm_oam_protocol_enable

This function enables OAM for the specified interface.

This function implements the `ethernet oam enable` command. The EFM MIB support also calls this function.

Syntax

```
s_int32_t
efm_oam_protocol_enable (struct interface *ifp)
```

Input Parameters

`ifp`	Pointer to the interface structure

Output Parameters

None

Return Values

EFM_OAM_ERR_IF_NOT_FOUND when the onmd interface is not found

EFM_OAM_ERR_ALREADY_ENABLED when OAM is already enabled

EFM_OAM_ERR_MEMORY when memory cannot be allocated

EFM_OAM_ERR_INTERNAL when there is an internal error

EFM_SUCCESS when the function succeeds

efm_oam_remote_loopback_get

This function gets the remote loopback setting.

SMI calls this function.

Syntax

```
#include "onmd/smi/smi_onm_server.h"
s_int32_t
efm_oam_remote_loopback_get (struct interface *ifp, u_int8_t *enable)
```

Input Parameters

`ifp`	Pointer to the interface structure

Output Parameters

`enable`	Whether remote-loopback support is enabled (1) or disabled (0)

Return Values

EFM_OAM_ERR_IF_NOT_FOUND when the onmd interface is not found

EFM_OAM_ERR_NOT_ENABLED when EFM is not enabled for the interface

EFM_OAM_ERR_INTERNAL when there is an internal error

EFM_SUCCESS when the function succeeds

efm_oam_remote_loopback_set

This function is called by the `ethernet oam remote-loopback supported` command.

SMI also calls this function.

Syntax

```
s_int32_t
efm_oam_remote_loopback_set (struct interface *ifp, u_int8_t enable)
```

Input Parameters

`ifp`	Pointer to the interface structure
`enable`	Whether to enable (PAL_TRUE) or disable (PAL_FALSE) remote-loopback support

Output Parameters

None

Return Values

EFM_OAM_ERR_IF_NOT_FOUND when the onmd interface is not found

EFM_OAM_ERR_NOT_ENABLED when EFM is not enabled for the interface

EFM_OAM_ERR_INTERNAL when there is an internal error

EFM_SUCCESS when the function succeeds

efm_oam_remoteloopback_start

This function starts remote loopback for the specified interface.

The function implements the `ethernet oam remote-loopback start` command.

Syntax

```
s_int32_t
efm_oam_remoteloopback_start (struct interface *ifp)
```

Input Parameters

`ifp`	Pointer to the interface structure

Output Parameters

None

Return Values

EFM_OAM_ERR_IF_NOT_FOUND when the onmd interface is not found

EFM_OAM_ERR_NOT_ENABLED when EFM is not enabled for the interface

EFM_OAM_ERR_REM_LB_NOT_SUPP when remote loopback is not supported

EFM_OAM_ERR_INTERNAL when there is an internal error

EFM_SUCCESS when the function succeeds

efm_oam_remoteloopback_stop

This function stops remote loopback for the specified interface.

The function implements the `ethernet oam remote-loopback stop` command.

Syntax

```
s_int32_t
efm_oam_remoteloopback_stop (struct interface *ifp)
```

Input Parameters

`ifp`	Pointer to the interface structure

Output Parameters

None

Return Values

EFM_OAM_ERR_IF_NOT_FOUND when the onmd interface is not found

EFM_OAM_ERR_NOT_ENABLED when EFM is not enabled for the interface

EFM_OAM_ERR_INTERNAL when there is an internal error

EFM_SUCCESS when the function succeeds

efm_oam_remote_loopback_timeout_get

This function gets the value of the remote loopback timeout.

SMI calls this function.

Syntax

```
s_int32_t
efm_oam_remote_loopback_timeout_get (struct interface *ifp, u_int8_t *timeout)
```

Input Parameters

`ifp`	Pointer to the interface structure

Output Parameters

`timeout`	Remote-loopback timeout value in seconds

Return Values

EFM_OAM_ERR_IF_NOT_FOUND when the onmd interface is not found

EFM_OAM_ERR_NOT_ENABLED when EFM is not enabled for the interface

EFM_OAM_ERR_INTERNAL when there is an internal error

EFM_SUCCESS when the function succeeds

efm_oam_remote_loopback_timeout_set

This function sets the remote loopback timeout interval.

The function implements the `ethernet oam remote-loopback timeout TIMER-VALUE` value command.

SMI also calls this function.

Syntax

```
s_int32_t
efm_oam_remote_loopback_timeout_set (struct interface *ifp, u_int8_t timeout)
```

Input Parameters

`ifp`	Pointer to the interface structure
`timeout`	Remote-loopback timeout value in seconds

Output Parameters

None

Return Values

EFM_OAM_ERR_IF_NOT_FOUND when the onmd interface is not found

EFM_OAM_ERR_NOT_ENABLED when EFM is not enabled for the interface

EFM_OAM_ERR_INTERNAL when there is an internal error

EFM_SUCCESS when the function succeeds

efm_oam_set_link_timer

This function sets the minimum number of PDUs that can be transmitted per second by an interface.

This function implements the `ethernet oam min-rate TIMER_VALUE` value command.

Syntax

```
s_int32_t
efm_oam_set_link_timer (struct interface *ifp, int secs)
```

Input Parameters

`ifp`	Pointer to the interface structure
`secs`	The minimum number of PDUs that can be transmitted per second

Output Parameters

None

Return Values

EFM_OAM_ERR_IF_NOT_FOUND when the onmd interface is not found

EFM_OAM_ERR_NOT_ENABLED when EFM is not enabled for the interface

EFM_OAM_ERR_INTERNAL when there is an internal error

EFM_SUCCESS when the function succeeds

efm_oam_set_loopback_ignore_rx

This function sets the status of the loopback ignore receive control of the local OAM client.

The EFM MIB support calls this function.

Syntax

```
s_int32_t
efm_oam_set_loopback_ignore_rx (struct interface *ifp, u_int8_t ignoreRxStatus)
```

Input Parameters

`ifp` Pointer to the interface structure

`ignoreRxStatus`

Ignore receive status; one of these constants from the enum lb_control in onmd/efm-oam/efm_oam.h:

`EFM_OAM_LB_REQUEST_IGNORE`

Ignore loopback requests

`EFM_OAM_LB_REQUEST_ACCEPT`

Accept loopback requests

Output Parameters

None

Return

EFM_FAILURE when the interface is null

EFM_OAM_ERR_IF_NOT_FOUND when the onmd interface is not found

EFM_OAM_ERR_NOT_ENABLED when EFM is not enabled for the interface

EFM_OAM_ERR_INTERNAL when there is an internal error

EFM_SUCCESS when the function succeeds

efm_oam_set_loopback_status

This function starts or ends loopback.

The EFM MIB support calls this function.

Syntax

```
s_int32_t
efm_oam_set_loopback_status(struct interface *ifp, u_int8_t lbStatus)
```

Input Parameters

`ifp` Pointer to the interface structure

`lbStatus` Loopback status; one of these constants from the efm_lb_status enum in onmd/efm-oam/efm_oam.h:

`EFM_OAM_INIT_LB`
Initializing loopback

`EFM_OAM_TERMINATING_LB`
Terminating loopback

Output Parameters

None

Return

EFM_FAILURE when the interface is null or the lbStatus parameter is invalid

EFM_OAM_ERR_IF_NOT_FOUND when the onmd interface is not found

EFM_OAM_ERR_NOT_ENABLED when EFM is not enabled for the interface

EFM_OAM_ERR_INTERNAL when there is an internal error

EFM_SUCCESS when the function succeeds

efm_oam_show_discovery

This function gets discovery information for the interface.

This function implements the `show ethernet oam discovery interface IFNAME` command.

Syntax

```
void
efm_oam_show_discovery (struct interface *ifp, structure cli *cli)
```


Input Parameters

`ifp`	Pointer to the interface structure

Output Parameters

`cli`	Pointer to the CLI structure

Return Values

None

efm_oam_show_event_log

This function gets an interface's event log.

The function implements the `show ethernet oam eventlog IFNAME range-start <1-100> range-end <1-100>` command.

Syntax

```
u_int32_t
efm_oam_show_event_log (struct cli *cli, struct interface *ifp, u_int32_t start_range,
                        u_int32_t end_range)
```

Input Parameters

`ifp`	Pointer to the interface structure
`start_range`	The line of the log on which to start printing
`end_range`	The line of the log on which to stop printing

Output Parameters

`cli`	Pointer to the CLI structure

Return Values

EFM_FAILURE when information is missing

EFM_SUCCESS when the information is printed successfully

efm_oam_show_interface

This function gets Ethernet OAM statistics.

The function implements the `show ethernet oam IFNAME` command.

Syntax

```
void
efm_oam_show_interface (struct interface *ifp, struct cli *cli);
```

Input Parameters

`ifp`	Pointer to the interface structure

Output Parameters

`cli`	Pointer to the CLI structure

Return Values

None

efm_oam_show_statistics

This function gets OAM statistics for the specified interface.

The function implements the `show ethernet oam statistics interface IFNAME` command.

Syntax

```
void
efm_oam_show_statistics (struct interface *ifp, struct cli *cli);
```

Input Parameters

`ifp`	Pointer to the interface structure

Output Parameters

`cli`	Pointer to the CLI structure

Return Values

None

efm_oam_show_status

This function gets the interface status.

The function implements the `show ethernet oam status` command.

Syntax

```
void
efm_oam_show_status (struct interface *ifp, struct cli *cli)
```

Input Parameters

`ifp`	Pointer to the interface structure

Output Parameters

`cli`	Pointer to the CLI structure

Return Values

None

efm_oam_sym_period_high_thres_get

This function gets the high word of the 8-octet threshold for the errored symbol period event.

The EFM MIB support calls this function.

Syntax

```
s_int32_t
```

```
efm_oam_sym_period_high_thres_get (struct interface *ifp, u_int32_t* thres_high)
```

Input Parameters

`ifp`	Pointer to the interface structure

Output Parameters

`thres_high`	The high word of the 8-octet threshold for the errored symbol period event

Return

EFM_FAILURE when the interface is null

EFM_OAM_ERR_IF_NOT_FOUND when the onmd interface is not found

EFM_OAM_ERR_NOT_ENABLED when EFM is not enabled for the interface

EFM_OAM_ERR_INTERNAL when there is an internal error

EFM_SUCCESS when the function succeeds

efm_oam_sym_period_high_thres_set

This function sets the high word of the 8-octet threshold for the errored symbol period event.

This function implements the `ethernet oam link-monitor threshold symbol-period high THRES_HIGH` command. The EFM MIB support also calls this function.

Syntax

```
s_int32_t
efm_oam_sym_period_high_thres_set (struct interface *ifp, u_int16_t thres_high)
```

Input Parameters

`ifp`	Pointer to the interface structure
`thres_high`	The high word of the 8-octet threshold for the errored symbol period event

Output Parameters

None

Return Values

EFM_OAM_ERR_IF_NOT_FOUND when the onmd interface is not found

EFM_OAM_ERR_NOT_ENABLED when EFM is not enabled for the interface

EFM_OAM_ERR_INTERNAL when there is an internal error

EFM_SUCCESS when the function succeeds

efm_oam_sym_period_high_window_get

This function gets the high word of the 8-octet window size for the errored symbol period event.

The EFM MIB support calls this function.

Syntax

```
s_int32_t
```

```
efm_oam_sym_period_high_window_get (struct interface *ifp, u_int32_t *window)
```

Input Parameters

`ifp`	Pointer to the interface structure

Output Parameters

`window`	The high word of the 8-octet window size for the errored symbol period event

Return

EFM_FAILURE when the interface is null

EFM_OAM_ERR_IF_NOT_FOUND when the onmd interface is not found

EFM_OAM_ERR_NOT_ENABLED when EFM is not enabled for the interface

EFM_OAM_ERR_INTERNAL when there is an internal error

EFM_SUCCESS when the function succeeds

efm_oam_sym_period_high_window_set

This function sets the high word of the 8-octet window size for the errored symbol period event.

The EFM MIB support calls this function.

Syntax

```
s_int32_t
efm_oam_sym_period_high_window_set (struct interface *ifp, u_int32_t window)
```

Input Parameters

`ifp`	Pointer to the interface structure
`window`	The high word of the 8-octet window size for the errored symbol period event

Output Parameters

None

Return

EFM_FAILURE when the interface is null

EFM_OAM_ERR_IF_NOT_FOUND when the onmd interface is not found

EFM_OAM_ERR_NOT_ENABLED when EFM is not enabled for the interface

EFM_OAM_ERR_INTERNAL when there is an internal error

EFM_SUCCESS when the function succeeds

efm_oam_sym_period_low_thres_get

This function gets the low word of the 8-octet threshold for the errored symbol period event.

The EFM MIB support calls this function.

Syntax

```
s_int32_t
efm_oam_sym_period_low_thres_get (struct interface *ifp, u_int32_t* thres_low)
```

Input Parameters

`ifp`	Pointer to the interface structure

Output Parameters

`thres_low`	The low word of the 8-octet threshold for the errored symbol period event

Return

EFM_FAILURE when the interface is null

EFM_OAM_ERR_IF_NOT_FOUND when the onmd interface is not found

EFM_OAM_ERR_NOT_ENABLED when EFM is not enabled for the interface

EFM_OAM_ERR_INTERNAL when there is an internal error

EFM_SUCCESS when the function succeeds

efm_oam_sym_period_low_thres_set

This function sets the low word of the 8-octet threshold for the errored symbol period event.

This function implements the `ethernet oam link-monitor symbol-period threshold low THRES_LOW` value command. The EFM MIB support also calls this function.

Syntax

```
s_int32_t
efm_oam_sym_period_low_thres_set (struct interface *ifp, u_int16_t thres_low)
```

Input Parameters

`ifp`	Pointer to the interface structure
`thres_low`	The low word of the 8-octet threshold for the errored symbol period event

Output Parameters

None

Return Values

EFM_OAM_ERR_IF_NOT_FOUND when the onmd interface is not found

EFM_OAM_ERR_NOT_ENABLED when EFM is not enabled for the interface

EFM_OAM_ERR_INTERNAL when there is an internal error

EFM_SUCCESS when the function succeeds

efm_oam_sym_period_low_window_get

This function gets the low word of the 8-octet window size for the errored symbol period event.

The EFM MIB support calls this function.

Syntax

```
s_int32_t
efm_oam_sym_period_low_window_get (struct interface *ifp, u_int32_t *window)
```

Input Parameters

`ifp`	Pointer to the interface structure

Output Parameters

`window`	The low word of the 8-octet window size for the errored symbol period event

Return

EFM_FAILURE when the interface is null

EFM_OAM_ERR_IF_NOT_FOUND when the onmd interface is not found

EFM_OAM_ERR_NOT_ENABLED when EFM is not enabled for the interface

EFM_OAM_ERR_INTERNAL when there is an internal error

EFM_SUCCESS when the function succeeds

efm_oam_sym_period_low_window_set

This function sets the low word of the 8-octet window size for the errored symbol period event.

The EFM MIB support calls this function.

Syntax

```
s_int32_t
efm_oam_sym_period_low_window_set (struct interface *ifp, u_int32_t window)
```

Input Parameters

`ifp`	Pointer to the interface structure
`window`	The low word of the 8-octet window size for the errored symbol period event

Output Parameters

None

Return

EFM_FAILURE when the interface is null

EFM_OAM_ERR_IF_NOT_FOUND when the onmd interface is not found

EFM_OAM_ERR_NOT_ENABLED when EFM is not enabled for the interface

EFM_OAM_ERR_INTERNAL when there is an internal error

EFM_SUCCESS when the function succeeds

efm_oam_sym_period_window_set

This function sets the size of the window for the errored symbol period event.

This function implements the `ethernet oam link-monitor symbol-period window WINDOW` command.

Syntax

```
s_int32_t
efm_oam_sym_period_window_set (struct interface *ifp, u_int16_t window)
```

Input Parameters

`ifp`	Pointer to the interface structure
`window`	The window for the errored symbol period event

Output Parameters

None

Return Values

EFM_OAM_ERR_IF_NOT_FOUND when the onmd interface is not found

EFM_OAM_ERR_NOT_ENABLED when EFM is not enabled for the interface

EFM_OAM_ERR_INTERNAL when there is an internal error

EFM_FAILURE when the interface is null

EFM_SUCCESS when the function succeeds

SNMP API

The Ethernet First Mile (EFM) SNMP support is implemented according to the RFC 4878 standard. The major features of Ethernet OAM relate to groups of MIB as follows:

- Discovery: Ethernet OAM control group (dot3OamControlGroup) and Ethernet OAM peer group (dot3OamPeerGroup)
- Remote loopback: Ethernet OAM loopback group (dot3OamLoopbackGroup)
- Link Monitoring: Ethernet OAM event configuration group (Dot3OamEventConfigEntry)
- Event Logging: Ethernet OAM event log group (Dot3OamEventLogEntry)

The sections below list the MIB objects in each of the groups and show how they relate to functions in the ZebOS-XP API. Click a function name to jump to that reference topic.

dot3OamControlGroup

Objects in this group control the local OAM entity. They also provide information such as the OAM functions and the maximum PDU size supported.

Object Type	Syntax	Access	Functions
dot3OamAdminState	INTEGER	read-write	`efm_oam_protocol_enable` `efm_oam_protocol_disable`
dot3OamOperStatus	INTEGER	read-only	`efm_oam_get_oper_status`

dot3OamMode	INTEGER	read-write	efm_oam_mode_active_set efm_oam_mode_passive_set efm_oam_mode_get
dot3OamMaxOamPduSize	Unsigned32	read-only	efm_oam_control_group_attribs
dot3OamConfigRevision	Unsigned32	read-only	
dot3OamFunctionsSupported	BITS	read-only	

dot3OamPeerGroup

Objects in this group contain information about the OAM peer for an Ethernet interface. OAM entities communicate with a single OAM peer entity on Ethernet links on which OAM is enabled and operating.

Object Type	Syntax	Access	Functions
dot3OamPeerMacAddress	MacAddress	read-only	efm_oam_peer_group_strings
dot3OamPeerVendorOui	EightOTwoOui	read-only	
dot3OamPeerVendorInfo	Unsigned32	read-only	efm_oam_peer_group_attribs
dot3OamPeerMode	INTEGER	read-only	
dot3OamPeerMaxOamPduSize	Unsigned32	read-only	
dot3OamPeerConfigRevision	Unsigned32	read-only	
dot3OamPeerFunctionsSupported	BITS	read-only	

dot3OamLoopbackGroup

Objects in this group control the loopback state of the local link and the status of the loopback function. Since loopback can be disruptive, dot3OamLoopbackIgnoreRx controls whether loopback requests that are received are processed or ignored.

Object Type	Syntax	Access	Functions
dot3OamLoopbackStatus	INTEGER	read-write	efm_oam_get_loopback_status efm_oam_set_loopback_status
dot3OamLoopbackIgnoreRx	INTEGER	read-write	efm_oam_get_loopback_ignore_rx efm_oam_set_loopback_ignore_rx

Dot3OamStatsEntry

Objects in this group contain counts of OAM PDUs transmitted and received on an Ethernet interface.

Object Type	Syntax	Access	Functions
dot3OamInformationTx	Counter32	read-only	efm_oam_get_stats_group_attribs
dot3OamInformationRx	Counter32	read-only	
dot3OamUniqueEventNotification-Tx	Counter32	read-only	
dot3OamUniqueEventNotification-Rx	Counter32	read-only	
dot3OamDuplicateEvent-NotificationTx	Counter32	read-only	
dot3OamDuplicateEvent-NotificationRx	Counter32	read-only	
dot3OamLoopbackControlTx	Counter32	read-only	
dot3OamLoopbackControlRx	Counter32	read-only	
dot3OamVariableRequestTx	Counter32	read-only	
dot3OamVariableRequestRx	Counter32	read-only	
dot3OamVariableResponseTx	Counter32	read-only	
dot3OamVariableResponseRx	Counter32	read-only	
dot3OamOrgSpecificTx	Counter32	read-only	
dot3OamOrgSpecificRx	Counter32	read-only	
dot3OamUnsupportedCodesTx	Counter32	read-only	
dot3OamUnsupportedCodesRx	Counter32	read-only	
dot3OamFramesLostDueToOam	Counter32	read-only	

Dot3OamEventConfigEntry

Objects in this group contain settings that enable event notifications and the thresholds that generate OAM events. The standard threshold crossing events are:

- Errored Symbol Period Event: the number of symbol errors exceeds a threshold within a given window defined by a number of symbols (for example, 1,000 symbols out of 1,000,000 had errors).
- Errored Frame Period Event: the number of frame errors exceeds a threshold within a given window defined by a number of frames (for example, 10 frames out of 1000 had errors).
- Errored Frame Event: the number of frame errors exceeds a threshold within a given window defined by a period of time (for example, 10 frames in 1 second had errors).

- Errored Frame Seconds Summary Event: the number of errored frame seconds exceeds a threshold within a given time period (for example, 10 errored frame seconds within the last 100 seconds). An errored frame second is defined as a 1 second interval which had one or more frame errors.

Attribute	Syntax	Access	Functions
dot3OamErrSymPeriod-WindowHi	Unsigned32	read-write	efm_oam_sym_period_high_window_set efm_oam_sym_period_high_window_get
dot3OamErrSymPeriod-WindowLo	Unsigned32	read-write	efm_oam_sym_period_low_window_set efm_oam_sym_period_low_window_get
dot3OamErrSymPeriod-ThresholdHi	Unsigned32	read-write	efm_oam_sym_period_high_thres_set efm_oam_sym_period_high_thres_get
dot3OamErrSymPeriod-ThresholdLo	Unsigned32	read-write	efm_oam_sym_period_low_thres_set efm_oam_sym_period_low_thres_get
dot3OamErrFramePeriod-Window	Unsigned32	read-write	efm_oam_err_frame_period_window_set efm_oam_err_frame_period_window_get
dot3OamErrFramePeriod-Threshold	Unsigned32	read-write	efm_oam_err_frame_period_low_thres_set efm_oam_err_frame_period_low_thres_get
dot3OamErrFrame-Window	Unsigned32	read-write	efm_oam_err_frame_window_set efm_oam_err_frame_window_get
dot3OamErrFrame-Threshold	Unsigned32	read-write	efm_oam_err_frame_low_thres_set efm_oam_err_frame_low_thres_get
dot3OamErrFrameSecs-SummaryWindow	Unsigned32	read-write	efm_oam_err_frame_second_window_set efm_oam_err_frame_second_window_get
dot3OamErrFrameSecs-SummaryThreshold	Unsigned32	read-write	efm_oam_err_frame_second_low_thres_set efm_oam_err_frame_second_low_thres_get
dot3OamErrFramePeriod-EvNotifEnable	TruthValue	read-write	efm_oam_err_event_notify_control_set efm_oam_err_event_notify_control_get
dot3OamErrSymPeriodEv-NotifEnable	TruthValue	read-write	
dot3OamErrFrameEv-NotifEnable	TruthValue	read-write	
dot3OamErrFrameSecs-EvNotifEnable	TruthValue	read-write	
dot3OamDyingGasp-Enable	TruthValue	read-write	
dot3OamCriticalEvent-Enable	TruthValue	read-write	

Dot3OamEventLogEntry

Objects in this group record events that have occurred at the Ethernet OAM level. These include local events and remote events.

Attribute	Syntax	Access
dot3OamEventLogIndex	Unsigned32	read-only
dot3OamEventLogTimestamp	TimeStamp	read-only
dot3OamEventLogOui	EightOTwoOui	read-only
dot3OamEventLogType	Unsigned32	read-only
dot3OamEventLogLocation	INTEGER	read-only
dot3OamEventLogWindowHi	Unsigned32	read-only
dot3OamEventLogWindowLo	Unsigned32	read-only
dot3OamEventLogThresholdHi	Unsigned32	read-only
dot3OamEventLogThresholdLo	Unsigned32	read-only
dot3OamEventLogValue	CounterBasedGauge64	read-only
dot3OamEventLogRunningTotal	CounterBasedGauge64	read-only
dot3OamEventLogEventTotal	Unsigned32	read-only

MIB Functions

This section describes functions related to reading and writing MIBs.

Function	Description
var_dot3OamEventConfigTable	Queries an SNMP variable in the event configuration table
var_dot3OamLoopbackTable	Queries an SNMP variable in the loopback table
var_dot3OamPeerTable	Queries an SNMP variable in the peer group table
var_dot3OamStatsTable	Queries an SNMP variable in the OAM statistics table
var_dot3OamTable	Queries an SNMP variable in the OAM table
write_dot3OamEventConfigTable	Sets an SNMP variable in the event configuration table
write_dot3OamLoopbackTable	Sets an SNMP variable in the loopback table
write_dot3OamLoopbackTable	Sets an SNMP variable in the OAM table

var_dot3OamEventConfigTable

This function is called when an external network management system queries an SNMP variable in the event configuration table.

Syntax

```
unsigned char *
var_dot3OamEventConfigTable(struct variable *vp, oid * name, size_t * length, int exact,
size_t * var_len, WriteMethod ** write_method, u_int32_t vr_id)
```

Input Parameters

`vp`	Pointer to SNMP variable structure
`name`	Pointer to the variable name (OID)
`length`	Number of elements (sub-ids) in the name
`exact`	Whether this request is a GET (exact match: PAL_TRUE) or a GETNEXT (PAL_FALSE)
`vr_id`	Unused

Output Parameters

`var_len`	Length of the variable that was read
`write_method`	Pointer to a pointer to the SET function (WriteMethod) for the variable

Return Value

A pointer to the value of the variable when the function succeeds

Zero (0) or NULL when the function fails

var_dot3OamLoopbackTable

This function is called when an external network management system queries an SNMP variable in the loopback table.

Syntax

```
unsigned char *
var_dot3OamLoopbackTable (struct variable *vp, oid * name, size_t * length, int exact,
size_t * var_len, WriteMethod ** write_method, u_int32_t vr_id)
```

Input Parameters

`vp`	Pointer to SNMP variable structure
`name`	Pointer to the variable name (OID)
`length`	Number of elements (sub-ids) in the name
`exact`	Whether this request is a GET (exact match: PAL_TRUE) or a GETNEXT (PAL_FALSE)
`vr_id`	Unused

Output Parameters

`var_len`	Length of the variable that was read
`write_method`	Pointer to a pointer to the SET function (WriteMethod) for the variable

Return Value

A pointer to the value of the variable when the function succeeds

Zero (0) or NULL when the function fails

var_dot3OamPeerTable

This function is called when an external network management system queries an SNMP variable in the peer group table.

Syntax

```
unsigned char *
var_dot3OamPeerTable(struct variable *vp, oid * name,
                     size_t * length, int exact,
                     size_t * var_len, WriteMethod ** write_method,
                     u_int32_t vr_id)
```

Input Parameters

`vp`	Pointer to SNMP variable structure
`name`	Pointer to the variable name (OID)
`length`	Number of elements (sub-ids) in the name
`exact`	Whether this request is a GET (exact match: PAL_TRUE) or a GETNEXT (PAL_FALSE)
`vr_id`	Unused

Output Parameters

`var_len`	Length of the variable that was read
`write_method`	Pointer to a pointer to the SET function (WriteMethod) for the variable

Return Value

A pointer to the value of the variable when the function succeeds

Zero (0) or NULL when the function fails

var_dot3OamStatsTable

This function is called when an external network management system queries an SNMP variable in the OAM statistics table.

Syntax

```
unsigned char *
var_dot3OamStatsTable(struct variable *vp, oid * name, size_t * length, int exact,
size_t * var_len, WriteMethod ** write_method, u_int32_t vr_id)
```

Input Parameters

`vp`	Pointer to SNMP variable structure
`name`	Pointer to the variable name (OID)
`length`	Number of elements (sub-ids) in the name

`exact`	Whether this request is a GET (exact match: PAL_TRUE) or a GETNEXT (PAL_FALSE)
`vr_id`	Unused

Output Parameters

`var_len`	Length of the variable that was read
`write_method`	Pointer to a pointer to the SET function (WriteMethod) for the variable

Return Value

A pointer to the value of the variable when the function succeeds

Zero (0) or NULL when the function fails

var_dot3OamTable

This function is called when an external network management system queries an SNMP variable in the OAM table.

Syntax

```
unsigned char *
var_dot3OamTable(struct variable *vp, oid * name, size_t * length, int exact,
size_t * var_len, WriteMethod ** write_method, u_int32_t vr_id)
```

Input Parameters

`vp`	Pointer to SNMP variable structure
`name`	Pointer to the variable name (OID)
`length`	Number of elements (sub-ids) in the name
`exact`	Whether this request is a GET (exact match: PAL_TRUE) or a GETNEXT (PAL_FALSE)
`vr_id`	Unused

Output Parameters

`var_len`	Length of the variable that was read
`write_method`	Pointer to a pointer to the SET function (WriteMethod) for the variable

Return Value

A pointer to the value of the variable when the function succeeds

Zero (0) or NULL when the function fails

write_dot3OamEventConfigTable

This function is called when an external network management system sets an SNMP variable in the event configuration table.

Syntax

```
int
write_dot3OamEventConfigTable(int action, u_char * var_val, u_char var_val_type,
size_t var_val_len, u_char * statP, oid * name, size_t name_len, struct variable *vp,
u_int32_t vr_id)
```


Input Parameters

action	Unused
var_val	Value of the variable
var_val_type	Data type of the variable; this constant from lib/asn1.h which corresponds to ASN.1's built-in simple type:
ASN_INTEGER	
var_val_len	Length of the variable
statP	Unused
name	Pointer to variable name (OID)
length	Length of the name
vp	Pointer to SNMP variable structure
vr_id	Unused

Output Parameters

None

Return Values

SNMP_ERR_NOERROR when the function succeeds

SNMP_ERR_GENERR when the function fails

SNMP_ERR_WRONGVALUE when var_val is invalid

SNMP_ERR_WRONGTYPE when var_val_type is invalid

SNMP_ERR_WRONGLENGTH when var_val_len is invalid

write_dot3OamLoopbackTable

This function is called when an external network management system sets an SNMP variable in the loopback table.

Syntax

```
int
write_dot3OamLoopbackTable (int action, u_char * var_val, u_char var_val_type,
size_t var_val_len, u_char * statP, oid * name, size_t name_len, struct variable *vp,
u_int32_t vr_id)
```

Input Parameters

action	Unused
var_val	Value of the variable
var_val_type	Data type of the variable; this constant from lib/asn1.h which corresponds to ASN.1's built-in simple type:
ASN_INTEGER	
var_val_len	Length of the variable
statP	Unused
name	Pointer to variable name (OID)
length	Length of the name

vp	Pointer to SNMP variable structure
vr_id	Unused

Output Parameters

None

Return Values

SNMP_ERR_NOERROR when the function succeeds

SNMP_ERR_GENERR when the function fails

SNMP_ERR_WRONGVALUE when var_val is invalid

SNMP_ERR_WRONGTYPE when var_val_type is invalid

SNMP_ERR_WRONGLENGTH when var_val_len is invalid

write_dot3OamTable

This function is called when an external network management system sets an SNMP variable in the OAM table.

Syntax

```
int
write_dot3OamTable(int action, u_char * var_val, u_char var_val_type,
size_t var_val_len, u_char * statP, oid * name, size_t name_len, struct variable *vp,
u_int32_t vr_id)
```

Input Parameters

action	Unused
var_val	Value of the variable
var_val_type	Data type of the variable; this constant from lib/asn1.h which corresponds to ASN.1's built-in simple type:
ASN_INTEGER	
var_val_len	Length of the variable
statP	Unused
name	Pointer to variable name (OID)
length	Length of the name
vp	Pointer to SNMP variable structure
vr_id	Unused

Output Parameters

None

Return Values

SNMP_ERR_NOERROR when the function succeeds

SNMP_ERR_GENERR when the function fails

SNMP_ERR_WRONGTYPE when var_val_type is invalid

SNMP_ERR_WRONGLENGTH when var_val_len is invalid

CHAPTER 11 Link Layer Discovery Protocol

The Link Layer Discovery Protocol (LLDP) as described in the specification IEEE 802.1AB 2005 is an agent running on an IEEE 802 LAN bridge that provides a mechanism for all the bridges connected to the LAN to send and receive connectivity and management related information to each other. The major capabilities of the system are sent to the neighboring stations using Link Layer Discovery Protocol Data Units (LLDPDU).

Note: To enable LLDPv2, LLDP-2005 should be disabled or vice versa.

Overview

The Link Layer Discover Protocol (LLDP) is a vendor-neutral Layer 2 protocol that allows a network device to advertise its identity and capabilities on a Local Area Network (LAN). The LLDP uses a type-length-value (TLV) to send and receive optional message information about devices. Type is a numerical code that indicates the kind of field represent by this part of the message. The length is the size of the value field, in bytes. Value is a variable-sized set of bytes that contains data for this part of the message.

LLDP is implemented as a separate protocol daemon and receives interface information through NSM to the client IPC.

LLDP MIB

LLDP maintains two Maintenance Information Bases (MIB):

1. Local LLDP MIB — Information retrieved from MIB implementation in NSM.
2. Remote LLDP MIB — Information retrieved from the LLDPDU received from remote LLDP client.

As implemented in ZebOS-XP, LLDP is part of the `onmd` daemon. The LLDP implementation includes both the protocol and the LLDP MIB module files. LLDP uses the NSM callback APIs to retrieve parameters for the Local LLDP MIB. The LLDP state machines work independently with a unique socket implementation. The LLDP Local MIB retrieves the information from the Protocol state machine and statistics variables and the TLV parameters stored. The Remote LLDP MIB retrieves the information from the Remote LLDPDU received and other LLDP Receive state machine and statistics variables. For detailed information about the LLDP MIB entities supported, refer to the *IEEE 802.1AB-2005 Station and Media Access Control Connectivity Discovery* specification, clause 12.

Operational Modes

ZebOS-XP LLDP supports these operational modes

- *Transmit-only mode.* The agent can only transmit information about the capabilities and current status of the local system.
- *Receive-only mode*. The agent can only receive information about the capabilities and current status of the local system.
- *Transmit and receive mode*. The agent can transmit the local system capabilities and status information, and it can receive the same information from a remote system.

LLDPDU Transmission and Reception

Transmission of LLDPDU is based upon one of these events

- Expiration of LLDP transmit countdown timer. To avoid frequent LLDPDU transmissions, a delay timer is set.

- Change in a local system parameter.

Reception of LLDPDU is predicated upon a valid destination and MAC Service Data Unit (MSDU) header combination.

LLDPDU Format

The LLDPDU frame format consists of the following fields

- Destination address: LLDP Multicast destination address is 01:80:c2:00:00:0e.
- Source address: The transmitting port's MAC address.
- Ethertype: The LLDP Ethertype is 88-CC.
- LLDPDU frame: The LLDPDU frame consists of the various mandatory and optional TLVs; the last TLV is the End of LLDPDU TLV.

Shutdown LLDPDU

A special LLDPDU is sent when a port is about to be disabled and is called the Shutdown LLDPDU. This frame consists of the following TLVs:

- Chassis ID TLV
- Port ID TLV
- TTL TLV with the TTL field value of zero
- End of LLDPDU TLV

Protocol Modules

The LLDP Agent runs on all the layer 2 ports of the bridge on which LLDP is enabled. The LLDP agent can only run on a physical port and not on any aggregated interface. The protocol state machines are implemented on per port basis.

Initialization

The protocol initialization module deals with initialization of default values for the various Protocol and state machine variables for each port. The port is being activated as an LLDP agent based on LLDP mode.

Transmit State Machine

The LLDP transmit state machine manages the construction and transmission of Normal and Shutdown LLDPDU. One LLDP transmit state machine is implemented per port.

Receive State Machine

The LLDP Receive State machine manages the reception, parsing, storing and aging out of LLDPDU from the Remote LLDP agent. One LLDP receive state machine is implemented per port.

Too Many Neighbors

The Too Many Neighbors state is used to control the size of the Remote LLDP MIB. The upper limit for the number of agents and the mode of handling is configurable. When the upper limit is reached:

- Received information — The Received LLDPDU is discarded.
- Existing information — The Remote LLDP MIB information for the configured agent is discarded.

TLV Support

For details about the Mandatory and Optional TLVs supported for LLDP, refer to the *IEEE 802.1AB-2005 Station and Media Access Control Connectivity Discovery* specification, clause 9.

LLDP Data Structures

The following are the LLDP functions as described in the specification IEEE 802.1AB 2005. These functions refer to the data structures as parameters.

lldp_master

This structure in `onmd/lldp/lldpd.h` defines the Link Layer Discover Protocol master.

Member name	Description
`onm`	omnd master
`sockfd`	Socket
`lldp_if_list`	LLDP interface list
`lldp_rcv_thread`	LLDP receive thread
`if_cnt`	Interface count
`syscap`	System capability
`sys_cap_enabled`	Enabled system capability
`lldp_stats_rem_drops`	LLDP port statistics frame drops
`lldp_stats_rem_inserts`	LLDP port statistics frame inserts
`lldp_stats_rem_deletes`	LLDP port statistics frame deletes
`sys_name`	System name
`sys_descr`	System description
`lldp_stats_rem_last_change_time`	LLPD statistics last change time
`lldp_dest_addr`	LLDP destination address
`mgmt_addr`	LLDP Management address
`conf_flag`	Configuration flag

```
struct lldp_master
{  struct onmd_master *onm;
  pal_sock_handle_t sockfd;
  struct thread *lldp_rcv_thread;
/*TODO - should be moved into TREE */
  struct list *lldp_if_list;
```

```
  u_int16_t  if_cnt;
  u_int16_t  syscap;
  u_int16_t  syscap_enabled;
  u_char sys_name[LLDP_NAME_MAX_SIZE + 1];
  u_char sys_descr[LLDP_DESCR_MAX_SIZE + 1];
  struct pal_in4_addr mgmt_addr;
/*Counters */
  pal_time_t lldp_stats_rem_last_change_time;
  u_int32_t lldp_stats_rem_tables_inserts;
  u_int32_t lldp_stats_rem_tables_deletes;
  u_int32_t lldp_stats_rem_tables_drops;
  u_int32_t lldp_state_rem_tables_ageouts;
#define LLDP_CONF_MGMT_IP_ADDRESS    (1 << 0)
  u_int8_t conf_flag;
};
```

LLDP Command API

This section contains the functions from LLDP that are called by the commands in the *Carrier Ethernet Command Line Interface Reference Guide*.

Function	Description
lldp_port_disable	Stops an LLDP agent
lldp_port_enable	Starts an LLDP agent
lldp_port_set_locally_assigned	Sets a string that identifies the port
lldp_set_port_basic_tlvs_enable	Sets the TLVs to be enabled for transmission
lldp_set_port_msg_tx_hold	Sets the TTL value for LLDPUs
lldp_set_port_msg_tx_interval	Sets the message transmit interval
lldp_set_port_reinit_delay	Sets the delay time between when LLDP is disabled on a port and an attempt to reinitialize it
lldp_set_port_too_many_neighbors	Sets the upper limit for Too Many Neighbors state in the remote LLDP
lldp_set_port_tx_delay	Sets the delay between successive transmissions of LLDP frame
lldp_set_system_description	Sets the string that describes the system
lldp_set_system_name	Sets the system name

Include Files

To call the functions in this section, you need to include these files:

- `onmd/lldp/lldpd.h`
- `onmd/lldp/lldp_api.h`

lldp_port_disable

This function is called to stop an LLDP agent on a port.

Syntax

```
#include "lldpd.h"
s_int32_t
lldp_port_disable (struct interface *ifp)
```

Input Parameters

`ifp`	Interface structure

Output Parameters

None

Return Values

LLDP_API_SUCCESS when the function succeeds

LLDP_API_ERR_LLDP_IF_NOT_EXIST when the interface is not found

LLDP_API_ERR_ONM_IF_NOT_EXIST when `onmd` is not found

lldp_port_enable

This function is called to start an LLDP agent on a port.

Syntax

```
#include "lldpd.h"
s_int32_t
lldp_port_enable (struct interface *ifp)
```

Input Parameters

`ifp`	Interface structure

Output Parameters

None

Return Values

LLDP_API_SUCCESS when the function succeeds

LLDP_API_ERR_INTERFACE_DOWN when the interface is down

LLDP_API_ERR_ONM_IF_NOT_EXIST when `onmd` is not found

lldp_port_set_locally_assigned

This function is called to locally assign the LLDP Port ID and the Chassis ID TLV parameters.

Syntax

```
#include "lldp_api.h"
```

```
s_int32_t
lldp_port_set_locally_assigned (struct interface *ifp, u_char *name)
```

Input Parameters

`ifp`	Interface structure
`name`	String that identifies the port

Output Parameters

None

Return Values

LLDP_API_SUCCESS when the function succeeds

LLDP_API_ERR_ONM_IF_NOT_EXIST when the `onmd` interface is not found

LLDP_API_ERR_LLDP_IF_NOT_EXIST when the LLDP interface is not found

lldp_set_port_basic_tlvs_enable

This function enables TLVs (type-length-values) to transmit on a port.

Syntax

```
#include "lldp_api.h"
s_int32_t
lldp_set_port_basic_tlvs_enable (struct interface *ifp, u_int16_t tlv_flag)
```

Input Parameters

`ifp`	Interface structure
`tlv_flag`	Flag identifying the TLVs to enable. One or more of the values below from `onmd/lldp/lldpd.h`. You can specify more than one value using bitwise or operators (“|”).

`CHASSIS_ID_TLV_TX_ENABLE`
: Chassis identification TLV

`PORT_ID_TLV_TX_ENABLE`
: Port identification TLV

`TTL_TLV_TX_ENABLE`
: Time to live TLV

`PORT_DESCRIPTION_TLV_TX_ENABLE`
: Port description TLV

`SYSTEM_NAME_TLV_TX_ENABLE`
: System name TLV

`SYSTEM_DESCRIPTION_TLV_TX_ENABLE`
: System description TLV

`SYSTEM_CAPABILITIES_TLV_TX_ENABLE`
: System capabilities TLV

`MANAGEMENT_ADDRESS_TLV_TX_ENABLE`

Management address TLV

`IEEE_8021_ORG_SPECIFIC_TLV_TX_ENABLE`

IEEE 802.1 organizationally specific TLVs: port VLAN ID, VLAN name, protocol VLAN ID, protocol ID

`IEEE_8023_ORG_SPECIFIC_TLV_TX_ENABLE`

IEEE 802.3 organizationally specific TLVs: MAC/PHY configuration/status, link aggregation, maximum frame size

Output Parameters

None

Return Values

LLDP_API_SUCCESS when the function succeeds

LLDP_API_ERR_ONM_IF_NOT_EXIST when `onmd` is not found

LLDP_API_ERR_LLDP_IF_NOT_EXIST when the LLDP interface is not found

LLDP_API_ERR_IF_PMIRROR_SET when the interface is mirrored

lldp_set_port_msg_tx_hold

This function sets the Message Transmit Hold parameter that determines to the TTL value for LLDPUs transmitted by the port.

Syntax

```
#include "lldp_api.h"
s_int32_t
lldp_set_port_msg_tx_hold (struct interface *ifp, u_int32_t value)
```

Input Parameters

`ifp`	Interface structure
`value`	Message transmit hold time <2-10>

Output Parameters

None

Return Values

LLDP_API_SUCCESS when the function succeeds

LLDP_API_ERR_ONM_IF_NOT_EXIST when the omnd interface is not found

LLDP_API_ERR_LLDP_IF_NOT_EXIST when the LLDP interface is not found

lldp_set_port_msg_tx_interval

This function is called to set the message transmit interval on the port.

Syntax

```
#include "lldp_api.h"
```

```
s_int32_t
lldp_set_port_msg_tx_interval (struct interface *ifp, u_int32_t value
```

Input Parameters

`ifp`	Interface structure
`value`	Message transmit interval <5-32768>

Output Parameters

None

Return Values

LLDP_API_SUCCESS when the function succeeds

LLDP_API_ERR_ONM_IF_NOT_EXIST when `onmd` is not found

LLDP_API_ERR_LLDP_IF_NOT_EXIST when the LLDP interface is not found

LLDP_API_ERR_MSG_TX_INTERVAL_ERR when the transmit interval is invalid

lldp_set_port_reinit_delay

This function sets the delay time between when LLDP is disabled on a port and an attempt to reinitialize it.

Syntax

```
#include "lldp_api.h"
s_int32_t
lldp_set_port_msg_tx_interval (struct interface *ifp, u_int32_t value)
```

Input Parameters

`ifp`	Interface structure
`value`	Reinitialize delay <1-10>

Output Parameters

None

Return Values

LLDP_API_SUCCESS when the function succeeds

LLDP_API_ERR_ONM_IF_NOT_EXIST when `onmd` is not found

LLDP_API_ERR_LLDP_IF_NOT_EXIST when the LLDP interface is not found

lldp_set_port_too_many_neighbors

This function sets the upper limit for Too Many Neighbors state in the Remote LLDP.

Syntax

```
#include "lldp_api.h"
s_int32_t
 lldp_set_port_too_many_neighbors (struct interface *ifp, u_int32_t limit,
                                   u_int8_t type, u_char *mac, u_int32_t interval)
```

Input Parameters

`ifp`	Interface structure
`limit`	The upper limit value for `too-many-neighbors`
`type`	Received LLDP information or existing LLDP information
`mac`	The MAC address of the Remote LLDP Agent
`interval`	Too-many-neighbors timer interval

Output Parameters

None

Return Values

LLDP_API_SUCCESS when the function succeeds

LLDP_API_ERR_ONM_IF_NOT_EXIST when `onmd` is not found

LLDP_API_ERR_LLDP_IF_NOT_EXIST when the LLDP interface is not found

LLDP_API_ERR_LLDP_MAC_ADDR_ERR when the MAC address is not valid

lldp_set_port_tx_delay

This function sets the delay between successive transmissions of LLDP frames.

Syntax

```
#include "lldp_api.h"
s_int32_t
lldp_set_port_tx_delay (struct interface *ifp, u_int32_t value)
```

Input Parameters

`ifp`	Interface structure
`value`	Transmit delay <1-8192>

Output Parameters

None

Return Values

LLDP_API_SUCCESS when the function succeeds

LLDP_API_ERR_ONM_IF_NOT_EXIST when `onmd` is not found

LLDP_API_ERR_LLDP_IF_NOT_EXIST when the LLDP interface is not found

LLDP_API_ERR_TX_DELAY_ERR when the transmit delay is not valid

lldp_set_system_description

This function is called to identify the string that describes the LLDP system.

Syntax

```
#include "lldp_api.h"
```

```
s_int32_t
lldp_set_system_description (struct lldp_master *lldpm, u_char *descr)
```

Input Parameters

`lldpm`	LLDP master structure
`descr`	String that describes the system

Output Parameters

None

Return Values

LLDP_API_ERR_LLDP_MASTER_NOT_EXIST when the LLDP master is not found

LLDP_API_SUCCESS when the function succeeds

lldp_set_system_name

This function is called to identify the system name of the LLDP function.

Syntax

```
#include "lldp_api.h"
s_int32_t
lldp_set_system_name (struct lldp_master *lldpm, u_char *name)
```

Input Parameters

`lldpm`	LLDP Master Structure
`name`	System name

Output Parameters

None

Return Values

LLDP_API_ERR_LLDP_MASTER_NOT_EXIST when the LLDP master is not found

LLDP_API_SUCCESS when the function succeeds

SNMP API

This section describes function that are called to support SNMP get and set operations.

Function	Description
lldp_snmp_set_msg_tx_interval	Sets the interval at which LLDP frames are transmitted
lldp_snmp_set_reinit_delay	Sets the time between when LLDP is disabled on the port and an attempt to reinitialize
lldp_snmp_set_tx_delay	Sets the delay between transmission of successive LLDP frames

Include File

To call the functions in this section, you need to include `onmd/lldp/lldp_api.h`.

lldp_snmp_set_msg_tx_interval

This function sets the interval at which LLDP frames are transmitted.

Syntax

```
s_int32_t
lldp_snmp_set_msg_tx_interval (u_int32_t msg_tx_interval)
```

Input Parameters

`msg_tx_interval` The interval

Output Parameters

None

Return Values

RESULT_OK when the function succeeds

RESULT_ERROR when the function fails

lldp_snmp_set_reinit_delay

This function sets the time between when LLDP is disabled on the port and an attempt to reinitialize.

Syntax

```
s_int32_t
lldp_snmp_set_reinit_delay(u_int32_t reinit_delay)
```

Input Parameters

`reinit_delay` The time between when LLDP is disabled on the port and an attempt to reinitialize

Output Parameters

None

Return Values

RESULT_OK when the function succeeds

RESULT_ERROR when the function fails

lldp_snmp_set_tx_delay

This function sets the `lldp tx delay` object.

Syntax

```
s_int32_t
lldp_snmp_set_tx_delay(u_int32_t tx_delay)
```

Input Parameters

`tx_delay`	The delay between transmission of successive LLDP frames

Output Parameters

None

Return Values

RESULT_OK when the function succeeds

RESULT_ERROR when the function fails

CHAPTER 12 Link Layer Discovery Protocol v2

The Link Layer Discovery Protocol v2 (LLDPv2), as described in the specification IEEE 802.1AB 2009 and LLDP-MED protocol extension as per ANSI/TIA-1057, are an agent running on an IEEE 802 LAN bridge that provides a mechanism for all the bridges connected to the LAN to send and receive connectivity and management related information to each other. The major capabilities of the system are sent to the neighboring stations using Link Layer Discovery Protocol Data Units (LLDPDU).

Note: To enable LLDPv2, LLDP-2005 should be disabled or vice versa.

Overview

The Link Layer Discover Protocol v2 (LLDPv2) is a vendor-neutral Layer 2 protocol that allows a network device to advertise its identity and capabilities on a Local Area Network (LAN). The LLDP uses a type-length-value (TLV) to send and receive optional message information about devices. Type is a numerical code that indicates the kind of field represent by this part of the message. The length is the size of the value field, in bytes. Value is a variable-sized set of bytes that contains data for this part of the message.

LLDP is implemented as a separate protocol daemon and receives interface information through NSM to the client IPC.

LLDPv2 Data Structures

The following are the LLDPv2 functions and these functions refer to the data structures as parameters.

lldp_rx_sm

This structure in `onmd/lldpv2/lldpd.h` is used to implement Rx state machine.

```
struct lldp_rx_sm
 {u_int8_t state;
  bool_t bad_frame;
  bool_t rx_changes;
  bool_t remote_changes;
  bool_t too_many_neighbours;
bool_t rx_info_age;
/* List of remote lldp parameters. */
  struct ptree *rlldp_list;
/*Timer Variables*/
 struct thread *too_many_neighbors_t;
/*Statistical Counters*/
  u_int32_t ageouts_total;
  u_int32_t frames_discarded_total;
  u_int32_t frames_in_errors_total;
  u_int32_t frames_in_total;
   u_int32_t tlvs_discarded_total;
  u_int32_t tlvs_unrecognized_total;

#define TOO_MANY_NEIGHBORS_ENABLED                    (1 << 0)
#define TOO_MANY_NEIGH_MAC_CONF                       (1 << 1)
```

```
  u_int8_t too_many_neighbors_flag;

#define TOO_MANY_NEIGHBORS_DISCARD_NONE                  0
#define TOO_MANY_NEIGHBORS_EXISTING_INFO                 1
#define TOO_MANY_NEIGHBORS_RECIEVED_INFO                 2
  u_int32_t too_many_neighbors_count;
  u_int32_t too_many_neighbors_limit;
#define TOO_MANY_NEIGHBORS_LIMIT_DEFAULT                 5
  u_int8_t too_many_neighbors_discard_type;

  u_char too_many_neighbors_discard_mac [ETHER_ADDR_LEN];
  u_int32_t too_many_neighbors_interval;
};
```

lldp_tx_sm

This structure in `onmd/lldpv2/lldpd.h` is used to implement Tx state machine.

```
struct lldp_tx_sm
{  #define LLDP_MSG_FAST_TX_DEFAULT                   1
  u_int16_t msg_fast_tx;
#define LLDP_MSG_TX_HOLD_DEFAULT                  4
  u_int16_t msg_tx_hold;
#define LLDP_MSG_TX_INTERVAL_DEFAULT              30
  u_int32_t msg_tx_interval;
#define LLDP_REINIT_DELAY_DEFAULT                 2
  u_int32_t reinit_delay;
#define LLDP_TX_CREDIT_MAX_DEFAULT                5
  u_int16_t tx_credit_max;
#define LLDP_TX_FAST_INIT_DEFAULT                 4
  u_int16_t tx_fast_init;
  u_int32_t tx_credit;
  u_int32_t tx_fast;
  u_int8_t state;
   bool_t tx_now;
   bool_t tx_tick;
   bool_t local_change;
    u_int16_t tx_ttl;
/* Timer variables */
  struct thread *tx_ttr_t;
  struct thread *tx_shutdown_while_t;
  struct thread *tx_one_sec_timer_t;

/*Statistical Counters*/
  u_int32_t frames_out_total;
  u_int32_t length_errors_total;
};
```

LLDPv2 Command API

This section contains the functions from LLDPv2 that are called by the commands in the *Carrier Ethernet Command Line Interface Reference Guide*.

Function	Description
lldp_api_add_agent	Start a LLDP agent mode
lldp_api_delete_agent	Stop a LLDP agent mode
lldp_api_set_agent_state	Enable/disable the LLDP capabilities on a LLDP agent port
lldp_api_set_max_credit	Set the maximum value of transmission
lldp_api_set_tx_fast_init	Set the LLDP frames that are transmitted
lldp_api_set_port_basic_tlvs_enable	Set the TLVs to be enabled for transmission
lldp_api_set_port_msg_tx_hold	Set the TTL value for LLDPUs
lldp_api_set_fast_tx_interval	Set the `msg-tx-hold` parameter
lldp_api_set_port_reinit_delay	set the interval at which LLDP frames are transmitted.
lldp_api_set_port_too_many_neighbors	Set the action to take when the remote table is full.
lldp_med_api_set_dev_type	configure the LLDP device type
lldp_med_api_select_tlv	Select the set of optional TLV's

Include Files

To call the functions in this section, you need to include these files:

- `onmd/lldpv2/lldpd.h`
- `onmd/lldpv2/lldp_api.h`

lldp_api_add_agent

This function is called to create a LLDP agent on a port.

Syntax

```
s_int32_t
lldp_api_add_agent (u_char *mac , struct interface *ifp)
s_int32_t
lldp_api_delete_agent (u_char *mac , struct interface *ifp)
```

Input Parameters

`mac` The MAC address of the LLDP Agent

`ifp`	Interface structure

Output Parameters

None

Return Values

LLDP_API_SUCCESS when the function succeeds

LLDP_API_ERR_LLDP_IF_NOT_EXIST when the interface is not found

LLDP_API_ERR_ONM_IF_NOT_EXIST when onmd is not found

lldp_api_delete_agent

This function is called to delete a LLDP agent on a port.

Syntax

```
s_int32_t
lldp_api_delete_agent (u_char *mac , struct interface *ifp)
```

Input Parameters

`mac`	The MAC address of the LLDP Agent
`ifp`	Interface structure

Output Parameters

None

Return Values

LLDP_API_SUCCESS when the function succeeds

LLDP_API_ERR_LLDP_IF_NOT_EXIST when the interface is not found

LLDP_API_ERR_ONM_IF_NOT_EXIST when `onmd` is not found

lldp_api_set_agent_state

This function is called to enable/disable the LLDP capabilities on a LLDP agent port.

Syntax

```
s_int32_t
lldp_api_set_agent_state (u_char *mac, struct interface *ifp, u_int8_t
admin_status)
```

Input Parameters

`mac`	The MAC address of the LLDP Agent
`ifp`	Interface structure

Output Parameters

None

Return Values

LLDP_API_SUCCESS when the function succeeds

LLDP_API_ERR_LLDP_IF_NOT_EXIST when the interface is not found

LLDP_API_ERR_ONM_IF_NOT_EXIST when `onmd` is not found

LLDP_API_ERR_AGENT_NOT_FOUND when agent is not found

lldp_api_set_fast_tx_interval

This function is called to set the interval at which LLDP frames are transmitted during fast transmission period.

Syntax

```
s_int32_t
lldp_api_set_fast_tx_interval(u_char *mac, struct interface *ifp,
                              u_int16_t interval)
```

Input Parameters

`mac`	The MAC address of the LLDP Agent
`ifp`	Interface structure

Output Parameters

None

Return Values

LLDP_API_SUCCESS when the function succeeds

LLDP_API_ERR_INVALID_VAL when the value is invalid

LLDP_API_ERR_TX_SM_NOT_FOUND

LLDP_API_ERR_LLDP_IF_NOT_EXIST when the interface is not found

LLDP_API_ERR_ONM_IF_NOT_EXIST when `onmd` is not found

LLDP_API_ERR_AGENT_NOT_FOUND when agent is not found

lldp_api_set_max_credit

This function is called to et the maximum value of transmission credit, which signifies the number of consecutive LLDP frames transmitted.

Syntax

```
s_int32_t
lldp_api_set_max_credit (u_char *mac, struct interface *ifp, u_int8_t credit)
```

Input Parameters

`mac`	The MAC address of the LLDP Agent
`ifp`	Interface structure

Output Parameters

None

Return Values

LLDP_API_SUCCESS when the function succeeds

LLDP_API_ERR_INVALID_VAL when the value is invalid

LLDP_API_ERR_TX_SM_NOT_FOUND

LLDP_API_ERR_LLDP_IF_NOT_EXIST when the interface is not found

LLDP_API_ERR_ONM_IF_NOT_EXIST when `onmd` is not found

LLDP_API_ERR_AGENT_NOT_FOUND when agent is not found

lldp_api_set_tx_fast_init

This function is called to determine the maximum value of LLDP frames that are transmitted during a fast transmission period.

Syntax

```
s_int32_t
lldp_api_set_tx_fast_init (u_char *mac, struct interface *ifp,
                           u_int8_t init_val)
```

Input Parameters

`mac`	The MAC address of the LLDP Agent
`ifp`	Interface structure

Output Parameters

None

Return Values

LLDP_API_SUCCESS when the function succeeds

LLDP_API_ERR_INVALID_VAL when the value is invalid

LLDP_API_ERR_TX_SM_NOT_FOUND

LLDP_API_ERR_LLDP_IF_NOT_EXIST when the interface is not found

LLDP_API_ERR_ONM_IF_NOT_EXIST when `onmd` is not found

LLDP_API_ERR_AGENT_NOT_FOUND when agent is not found

lldp_api_set_port_basic_tlvs_enable

This function is called to select the set of optional TLV's to be included in the LLDP frames.

Syntax

```
s_int32_t
lldp_api_set_port_basic_tlvs_enable (u_char *mac, struct interface *ifp,
               u_int16_t tlv_flag, u_int16_t opt_tlv_flag)
```


Input Parameters

`mac`	The MAC address of the LLDP Agent
`ifp`	Interface structure
`tlv_flag`	Flag identifying the TLVs to enable. One or more of the values below from `onmd/lldp/lldpd.h`. You can specify more than one value using bitwise or operators (“|”).

Output Parameters

None

Return Values

LLDP_API_SUCCESS when the function succeeds

LLDP_API_ERR_INVALID_VAL when the value is invalid

LLDP_API_ERR_LLDP_IF_NOT_EXIST when the interface is not found

LLDP_API_ERR_ONM_IF_NOT_EXIST when `onmd` is not found

LLDP_API_ERR_AGENT_NOT_FOUND when agent is not found

lldp_api_set_port_msg_tx_hold

This function is called to set the `msg-tx-hold` parameter that determines the Time To Live (TTL) value for LLDPDUs to be transmitted by the port. The value set with this command is multiplied by the `lldp timer msg-tx-interval` value, which determines the final TTL value.

Syntax

```
s_int32_t
lldp_api_set_port_msg_tx_hold (u_char* mac, struct interface *ifp,
                               u_int32_t value)
```

Input Parameters

`mac`	The MAC address of the LLDP Agent
`ifp`	Interface structure

Output Parameters

None

Return Values

LLDP_API_SUCCESS when the function succeeds

LLDP_API_ERR_INVALID_VAL when the value is invalid

LLDP_API_ERR_LLDP_IF_NOT_EXIST when the interface is not found

LLDP_API_ERR_ONM_IF_NOT_EXIST when `onmd` is not found

LLDP_API_ERR_AGENT_NOT_FOUND when agent is not found

lldp_api_set_port_msg_tx_interval

This function is called to set the interval at which LLDP frames are transmitted.

Syntax

```
s_int32_t
lldp_api_set_port_msg_tx_interval (u_char *mac, struct interface *ifp,
                                   u_int32_t value)
```

Input Parameters

`mac`	The MAC address of the LLDP Agent
`ifp`	Interface structure

Output Parameters

None

Return Values

LLDP_API_SUCCESS when the function succeeds

LLDP_API_ERR_INVALID_VAL when the value is invalid

LLDP_API_ERR_LLDP_IF_NOT_EXIST when the interface is not found

LLDP_API_ERR_ONM_IF_NOT_EXIST when `onmd` is not found

LLDP_API_ERR_AGENT_NOT_FOUND when agent is not found

lldp_api_set_port_reinit_delay

This function is called to set the interval at which LLDP frames are transmitted.

Syntax

```
lldp_api_set_port_reinit_delay (u_char *mac, struct interface *ifp,
                                u_int32_t value)
```

Input Parameters

`mac`	The MAC address of the LLDP Agent
`ifp`	Interface structure

Output Parameters

None

Return Values

LLDP_API_SUCCESS when the function succeeds

LLDP_API_ERR_INVALID_VAL when the value is invalid

LLDP_API_ERR_LLDP_IF_NOT_EXIST when the interface is not found

LLDP_API_ERR_ONM_IF_NOT_EXIST when `onmd` is not found

LLDP_API_ERR_AGENT_NOT_FOUND when agent is not found

lldp_api_set_port_too_many_neighbors

This function is called to set the action to take when the remote table is full.

Syntax

```
s_int32_t
lldp_api_set_port_too_many_neighbors (u_char * mac, struct interface *ifp,
                                      u_int32_t limit,
                                      u_int8_t type, u_char *discard_mac,
                                      u_int32_t interval)
```

Input Parameters

`mac`	The MAC address of the LLDP Agent
`ifp`	Interface structure

Output Parameters

None

Return Values

LLDP_API_SUCCESS when the function succeeds

LLDP_API_ERR_INVALID_VAL when the value is invalid

LLDP_API_ERR_LLDP_IF_NOT_EXIST when the interface is not found

LLDP_API_ERR_ONM_IF_NOT_EXIST when `onmd` is not found

LLDP_API_ERR_AGENT_NOT_FOUND when agent is not found

lldp_med_api_set_dev_type

This function is called to configure the LLDP device type as Network-Connectivity/ End-Point Class1/ End-Point Class2/End-Point Class3 device.

Syntax

```
s_int32_t
lldp_med_api_set_dev_type (u_char *mac, struct interface *ifp
                           enum med_dev_type dev_type)
```

Input Parameters

`mac`	The MAC address of the LLDP Agent
`ifp`	Interface structure

Output Parameters

None

Return Values

LLDP_API_SUCCESS when the function succeeds

LLDP_API_ERR_INVALID_VAL when the value is invalid

LLDP_API_ERR_LLDP_IF_NOT_EXIST when the interface is not found

LLDP_API_ERR_ONM_IF_NOT_EXIST when `onmd` is not found

LLDP_API_ERR_AGENT_NOT_FOUND when agent is not found

lldp_med_api_select_tlv

This function is called to select the set of optional TLV's which can enabled for transmission.

Syntax

s_int32_t
```
lldp_med_api_select_tlv (u_char *mac , struct interface *ifp,
                         u_int16_t tlv_flag)
```

Input Parameters

`mac`	The MAC address of the LLDP Agent
`ifp`	Interface structure
`tlv_flag`	Flag identifying the TLVs to enable. One or more of the values below from `onmd/lldp/lldpd.h`. You can specify more than one value using bitwise or operators ("|").

Output Parameters

None

Return Values

LLDP_API_SUCCESS when the function succeeds

LLDP_API_ERR_INVALID_VAL when the value is invalid

LLDP_API_ERR_LLDP_IF_NOT_EXIST when the interface is not found

LLDP_API_ERR_ONM_IF_NOT_EXIST when `onmd` is not found

LLDP_API_ERR_AGENT_NOT_FOUND when agent is not found

SNMP API

This section describes function that are called to support SNMP get and set operations.

Function	Description
lldpv2_snmp_set_msg_tx_interval	Sets the interval at which LLDP frames are transmitted
lldpv2_snmp_set_port_msg_tx_hold	Sets the value of message transmit hold parameter
lldpv2_snmp_set_msg_fast_tx	Sets the interval at which LLDP frames are transmitted during the fast transmission period
lldpv2_snmp_set_port_reinit_delay	Sets the delay value for 'reinit delay'
lldpv2_snmp_set_max_credit	Sets the maximum number of consecutive LLDPDUs that can be transmitted at any time
lldpv2_snmp_set_tx_fast_init	Sets the variable which determines the number of LLDPDUs that are transmitted during the transmission mode
lldpv2_snmp_set_tlvs	Specifies the basic set of LLDP TLVs
lldpv2_snmp_set_admin_status	Sets the administratively desired status of the local LLDP agent

Function	Description
lldpv2_snmp_set_ntfy_interval	Sets the interval between transmissions of LLDP notifications during normal transmission periods
lldpv2_snmp_set_ntfy_enable	Configures whether or not notifications from the agent are enabled
lldpv2_snmp_get_tlvs	Returns the basic set of LLDP TLVs whose transmissions are allowed on the local LLDP
lldpv2_snmp_get_admin_status	Returns the administratively desired status of the local LLDP agent

Include File

To call the functions in this section, you need to include `onmd/lldp/lldp_api.h`.

lldpv2_snmp_set_msg_tx_interval

This function sets the interval at which LLDP frames are transmitted.

Syntax

```
s_int32_t
lldpv2_snmp_set_msg_tx_interval (u_int32_t msg_tx_interval)
```

Input Parameters

`msg_tx_interval` Message transmit interval <5-32768>

Output Parameters

None

Return Values

LLDP_API_SUCCESS when the function succeeds

LLDP_API_ERROR when the function fails

lldpv2_snmp_set_port_msg_tx_hold

This function sets the message transmit hold parameter.

Syntax

```
s_int32_t
lldpv2_snmp_set_port_msg_tx_hold (u_int32_t  value )
```

Input Parameters

`value` Message transmit hold time <2-10>

Output Parameters

None

Return Values

LLDP_API_SUCCESS when the function succeeds
LLDP_API_ERROR when the function fails

lldpv2_snmp_set_msg_fast_tx

This function set the interval at which LLDP frames are transmitted during the fast transmission period.

Syntax

```
s_int32_t
lldpv2_snmp_set_msg_fast_tx (u_int32_t  value )
```

Input Parameters

`value`	Specify the message transmit value.

Output Parameters

None

Return Values

LLDP_API_SUCCESS when the function succeeds
LLDP_API_ERROR when the function fails

lldpv2_snmp_set_port_reinit_delay

This function is called to set the delay value for ‘reinit delay’. This delay value indicates the delay from when `PortConfigAdminStatus` object of a particular port becomes 'disabled' until re-initialization is attempted.

Syntax

```
s_int32_t
lldpv2_snmp_set_port_reinit_delay (u_int32_t  value )
```

Input Parameters

`value`	Specify the reinit delay value.

Output Parameters

None

Return Values

LLDP_API_SUCCESS when the function succeeds
LLDP_API_ERROR when the function fails

lldpv2_snmp_set_max_credit

This function is called to set the maximum number of consecutive LLDPDUs that can be transmitted at any time.

Syntax

```
s_int32_t
lldpv2_snmp_set_max_credit(u_int32_t  credit )
```

Input Parameters

`credit` Specify the number of LLDPDU to be transmitted.

Output Parameters

None

Return Values

LLDP_API_SUCCESS when the function succeeds
LLDP_API_ERROR when the function fails

lldpv2_snmp_set_tx_fast_init

This function is called to set the variable which determines the number of LLDPDUs that are transmitted during the transmission mode.

Syntax

```
s_int32_t
lldpv2_snmp_set_tx_fast_init(u_int32_t  initval )
```

Input Parameters

`initval` Specify the number of LLDPDU to be transmitted.

Output Parameters

None

Return Values

LLDP_API_SUCCESS when the function succeeds
LLDP_API_ERROR when the function fails

lldpv2_snmp_set_tlvs

This function is called to specify the basic set of LLDP TLVs whose transmissions are allowed on the local LLDP agent.

Syntax

```
s_int32_t
lldpv2_snmp_set_tlvs (u_charval struct lldp_agent *lldp_ag)
```

Input Parameters

`lldp_agent *lldp_ag`

Specify the set basic LLDP TLV

Output Parameters

None

Return Values

LLDP_API_SUCCESS when the function succeeds
LLDP_API_ERROR when the function fails

lldpv2_snmp_set_admin_status

This function is called to set the administratively desired status of the local LLDP agent.

Syntax

```
s_int32_t
lldpv2_snmp_set_admin_status(u_int32_t val struct lldp_agent *lldp_ag)
```

Input Parameters

`lldp_agent *lldp_ag`

Specify the pointer to the LLDP agent structure

Output Parameters

None

Return Values

LLDP_API_SUCCESS when the function succeeds
LLDP_API_ERROR when the function fails

lldpv2_snmp_set_ntfy_interval

This function is called to set the interval between transmissions of LLDP notifications during normal transmission periods.

Syntax

```
s_int32_t
lldpv2_snmp_set_ntfy_interval(u_int32_t ntfy_interval)
```

Input Parameters

`ntfy_interval` Specify the notification interval value

Output Parameters

None

Return Values

LLDP_API_SUCCESS when the function succeeds
LLDP_API_ERROR when the function fails

lldpv2_snmp_set_ntfy_enable

This function is called to configure whether or not notifications from the agent are enabled.

Syntax

```
s_int32_t
lldpv2_snmp_set_ntfy_interval(u_int32_t ntfy struct lldp_agent *lldp_ag)
```

Input Parameters

`ntfy` Specify the notification status

`lldp_agent *lldp_ag`

Specify the pointer to the LLDP agent structure

Output Parameters

None

Return Values

LLDP_API_SUCCESS when the function succeeds
LLDP_API_ERROR when the function fails

lldpv2_snmp_get_tlvs

This function returns the basic set of LLDP TLVs whose transmissions are allowed on the local LLDP.

Syntax

```
s_int32_t
lldpv2_snmp_get_tlvs(struct lldp_agent *lldp_ag)
```

Input Parameters

`lldp_agent *lldp_ag`

Specify the pointer to the LLDP agent structure

Output Parameters

None

Return Values

`<s_int32_t>` Returns the bitmap about basic set of LLDP TLVs whose transmissions are allowed.

lldpv2_snmp_get_admin_status

This function returns the administratively desired status of the local LLDP agent.

Syntax

```
s_int32_t
lldpv2_snmp_get_admin_status(struct lldp_agent *lldp_ag)
```

Input Parameters

`lldp_agent *lldp_ag`

Specify the pointer to the LLDP agent structure

Output Parameters

None

Return Values

`<s_int32_t>` Returns the administrative status of the local LLDP agent

CHAPTER 13 User Network Interface

ZebOS-XP User Network Interface (UNI) provides a demarcation point between the responsibility of the service provider and the responsibility of subscribers. It offers a reference point for Ethernet service delivery by establishing a standards-based connection through a Metro Ethernet Network (MEN).

Overview

Physically implemented over a bidirectional Ethernet link, UNI provides a variety of data, control and management plane capabilities required by MEN service providers. The functionality of UNI is to separate clearly the subscribers of two network domains, which are involved in the operational, administrative, maintenance and provisioning aspects of their own network services. Figure 13-1 on page 317 displays a simple diagram of an UNI operation.

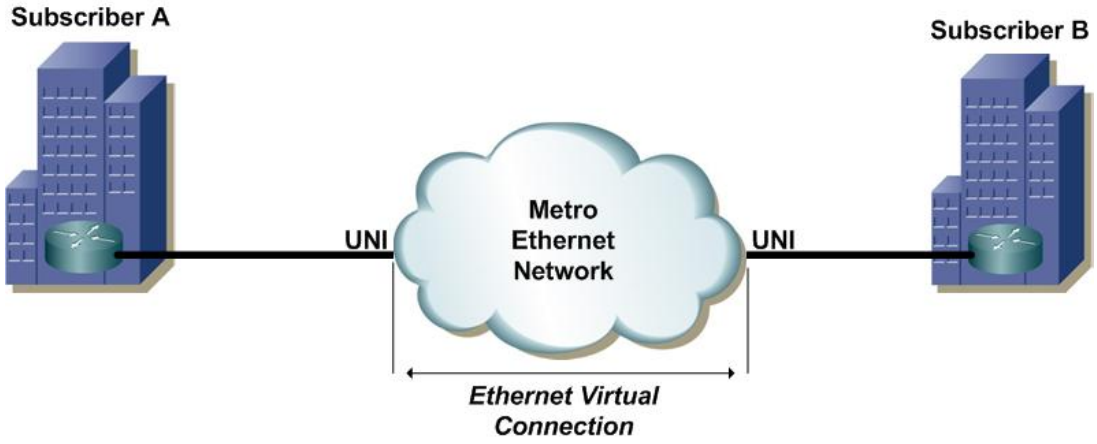


Figure 13-1: UNI Operation

Features

The following are some of the features of UNI:

- Full implementation and support for the following:
  - Section 10 of MEF 11
  - Sections 8, 9, 10 of MEF 20
  - Sections 6, 7 and 8.4 of MEF-10.1h MEF 13 (UNI-Type1) and MEF20 (UNI-Type 2) UNI specifications
- Provides a single-point of contact that is the demarcation between customers and service providers
- Located on a port of an active device that is operated by the service provider
- Easily integrated with other MEF defined functions
- Automatically configures and manages Ethernet UNI functions and services
- Complete OAM functions, including support for Connectivity Fault Management (CFM)
- Increased bandwidth and load sharing with linear increments of units of links

UNI Reference Point

Theoretically, the User Network Interface is an abstract concept, with the service provider responsible for one side of the configuration and the customer responsible for the other side. The demarcation of these two points divides at the reference point of UNI. This point helps service the Ethernet Virtual Connections (EVCs) that connect each UNI. Typically, the reference point for UNI is located at the premises of the subscriber and acts as the demarcation point between the subscriber and the MEN boundaries. It utilizes the connector that the subscriber network uses to connect the service provider network. Functionally, the reference point for UNI divides the role of the subscriber (known as UNI-C) and the role of the MEN (known as UNI-N).

- UNI-N is responsible for all aspects of UNI from the Provider Edge (PE) to the reference point for UNI. Moreover, UNI-N refers to a set of functional elements that supports the technical capabilities and compliance to the specification for UNI from the MEN service provider.
- UNI-C is responsible for all aspects of UNI from the Customer Edge (E-LMI) to the reference point of UNI. Moreover, UNI-C refers to the functional elements within the E-LMI that supports the technical capabilities and compliance to the specification for UNI from the MEN subscriber.

Because of the distributed nature of UNI, maintenance of both UNI-C and UNI-N is performed by different autonomies.

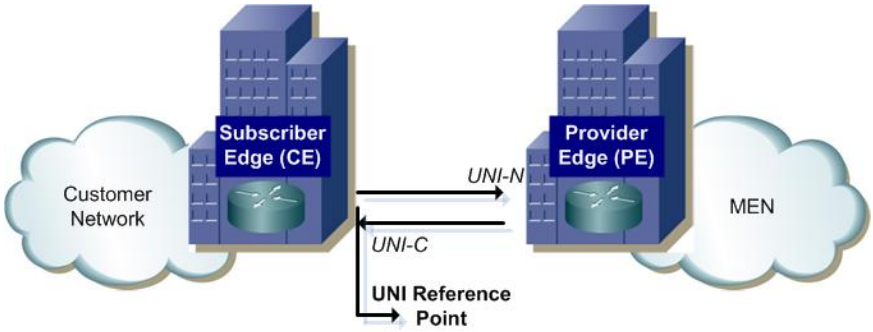


Figure 13-2: UNI Detailed Operation

UNI Functional Elements

UNI includes two functional elements that are located at the connected devices at either side of the reference point of UNI (demarcation): UNI-C and UNI-N.

- UNI-C: Responsible to execute all of the processes from the customer side
- UNI-N: Responsible to execute all of the processes from the network side

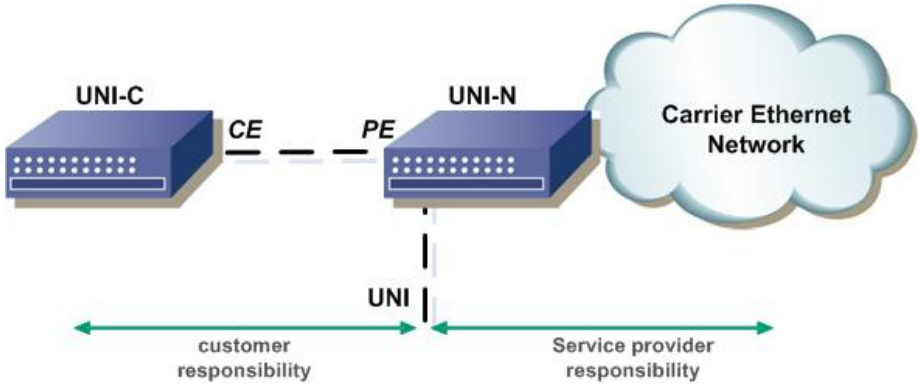


Figure 13-3: UNI Functional Elements

UNI-C and UNI-N Relationship

The following list describes the relationships for UNI-C and UNI-N among the service frames (that is, user generated), control and Carrier Ethernet management frames.

- Data plane functional elements for both UNI-C and UNI-N processes the subscriber-to-subscriber service frames (including the data, control and management frames)
- Control plane functional elements process for both UNI-C and UNI-N processes the control frames between the subscriber and the service provider
- Management plane functional elements for both UNI-C and UNI-N process the management frames between the subscriber and the service provider

UNI-Types

UNI framework supports two types of UNI requirements: UNI-Type 1 and UNI-Type 2.

UNI-Type 1

As defined in the MEF 13 specifications, UNI-Type 1 mode allows service providers and customers to configure manually both UNI-N and UNI-C networks for services. UNI-Type 1 helps Ethernet deployment of existing customer equipment, for example, networks that require no changes and use existing IEEE Ethernet physical layer and MAC functionality.

UNI-Type 2

As defined in the MEF 20 specifications, UNI-Type 2 mode allows UNI-C to retrieve EVC status and configuration information from UNI-N. In addition, UNI-Type 2 adds fault management beyond that specified in UNI-Type 1. UNI-Type 2 also provides UNI with service discovery functionality. This functionality supports automatic service discovery and OAM to allow a UNI-C network to provision automatically by discovering service attributes across UNI. Moreover, the OAM functionality provides mechanisms that manage and troubleshoot UNI. UNI-TYPE 2 divides into two categories: UNI-Type 2.1 and UNI-Type 2.2.

UNI-Type 2.1

UNI-Type 2.1 interface supports OAM (Operations, Administration & Maintenance) services, including CFM (Connectivity Fault Management). LACP (Link Aggregation Control Protocol) and EFM (Ethernet in the First Mile) are considered operational by default. UNI-Type 2.1 supports the service OAM protocol and is active; however, the E-LMI protocol is optional. If the E-LMI protocol is operational, then the type is UNI-Type 2.2. If the E-LMI protocol is not operational and if CFM is operational, then the type is UNI-Type 2.1.

UNI-Type 2.2

UNI-Type 2.2 type supports OAM, E-LMI and LINK OAM services. For E-LMI, UNI-Type 2.2 must be present and active. This means that E-LMI must be operational when UNI-Type is UNI-Type 2.2. The presence of the service OAM protocol is sufficient and could possibly be operational. In addition, UNI-Type 2.2 supports the LINK OAM protocol; however, it does not have to be active. By default, LINK OAM is considered UP, since the user is able to configure LACP or EFM to support these protocols.

Backward Compatibility

UNI-Type 2 UNI-C and UNI-N are backward compatible with UNI-Type 1. Backward compatibility means that UNI-Type 2 must continue to support all UNI-Type-1 requirements. As per MEF 11, a Type 2 UNI-N must automatically detect a Type 1 UNI-C and fall back to UNI-Type 1. Similarly, a Type 2 UNI-C must automatically detect a Type 1 UNI-N and fall back to UNI-Type 1. If SERVICE OAM and ELMI functionality of UNI-Type 2 are not operational, then the type will fall back to UNI-Type 1.0. If the subset of UNI-Type 2 is not operational, then type will be UNI-Type 2 but UNI-Type-2 mode

can be type 2.1 or type 2.2. As per MEF 20 section 8, UNI-Type 2 discovery and configuration functionality determines the operational status of CFM, and E-LMI at the remote end.

Ethernet Local Management Interface

1. In E-LMI, a UNI-N uses the polling verification timer to determine the operational status of E-LMI at UNI-C and send a message to NSM regarding the operational status of E-LMI.
2. A UNI-C uses the polling timer to determine the operational status of E-LMI at UNI-N and sends this message to NSM.

E-LMI operational status must be active for the Type 2.2. If it is not operational, then the Type 2 mode changes to UNI-Type 2.1 if Service OAM (SOAM) is operational. The type falls back to UNI-Type 1.0 if both E-LMI and SOAM are not operational.

Service OAM

The Service OAM is supported as per the IEEE-802.1ag 2007 and ITUT Y1731 standards. UNI MEP configuration and convergence determines the operational status of CFM at UNI-N or UNI-C.

1. Users must configure UNI-N with the remote MEP of a UNI-C MEP-ID and configure UNI-C with the remote MEP of UNI-N MEP ID.
2. If configured, UNI-MEP enables the CCMs by default and the status of UNI-MEP determines the operational status of CFM at the remote end.

If E-LMI is operational, then the type is UNI-Type 2.2, regardless of the operational status of CFM. If E-LMI is not operational, then the operational status of CFM is checked to decide the type of UNI.

Processing at NSM

Any change in the operational status of any of the above protocols triggers NSM, which is maintained in the interface structure of NSM. If UNI-type mode is enabled, then the type of UNI is determined based on the change of the operational status. Otherwise, UNI-type does not update, but the operational status of E-LMI and CFM are stored at NSM. Use the show command of UNI-type to display the operational status of these two protocols.

Link OAM

By default, the operational status of LACP and EFM is up. The support of EFM and LACP are present by default, but the operation of the protocols is optional. User may not enable EFM and CFM at the same time and on the same link.

Ethernet Service Attributes

UNI-Type 1 supports the Ethernet service attributes as described in the MEF 10.1 standard. Ethernet service attributes include EVC service attributes per UNI and EVC service attributes, E-LMI, Link OAM, and Service OAM.

UNI and EVC Attributes

Table 13-1 list the attributes for UNI and EVC supported by UNI.

Table 13-1: UNI and EVC Attributes

Attribute	Parameter
UNI Identifier	Any string with max size of 64 bytes
Physical Medium	A Standard Ethernet PHY
Speed	10 Mbps, 100 Mbps, 10/100 Mbps Auto-Negotiation, 1 Gbps, or 10 Gbps
Mode	Full-duplex
MAC Layer	EEE 802.3 - 2005
UNI MTU Size	Integer ≥ 1522
Service Multiplexing	Yes or No
UNI EVC ID	A string formed by the concatenation of UNI ID and the EVC ID
CE-VLAN ID for untagged and priority tagged Service Frames	A number in 1, 2… 4094
CE-VLAN ID/EVC Map	CVLAN to SVLAN mapping
Maximum Number of EVCs	Integer ≥ 1
Bundling	Yes or No
All to One Bundling	Yes or No
Ingress Bandwidth Profile Per Ingress UNI	No or BW profile parameters
Ingress Bandwidth Profile Per EVC	No or BW profile parameters
Ingress Bandwidth Profile Per Class of Service Identifier	No or BW profile parameters
Egress Bandwidth Profile Per Egress UNI	No or BW profile parameters
Egress Bandwidth Profile Per EVC	No or BW profile parameters
Egress Bandwidth Profile Per Class of Service Identifier	No or BW profile parameters
Layer 2 Control Protocols Processing	A list of Layer 2 Control Protocols with each being labeled with one of Discard, Peer, Pass to EVC, Peer and Pass to EVC

EVC Attributes

Table 13-2 list some of the EVC attributes supported by UNI.

Table 13-2: EVC Attributes

Attribute	Parameter
EVC Type	Point-to-point or multipoint-to-multipoint

Table 13-2: EVC Attributes (Continued)

Attribute	Parameter
EVC ID	An arbitrary string, unique across the MEN, for the EVC supporting the service instance.
UNI List	A list of <UNI Identifier, UNI-Type> pairs
Maximum Number of UNIs	Integer. MUST be 2 if EVC Type is Point-to-Point. MUST be greater than or equal to 2 otherwise.
EVC Maximum Transmission Unit Size	Integer ≥ 1522.
CE-VLAN ID Preservation	Yes or No
CE-VLAN CoS Preservation	Yes or No
Unicast Service Frame Delivery	Discard, Deliver Unconditionally, or Deliver Conditionally. If Deliver Conditionally is used, then conditions must be specified.
Multicast Service Frame Delivery	Discard, Deliver Unconditionally, or Deliver Conditionally. If Deliver Conditionally is used, then conditions must be specified.
Broadcast Service Frame Delivery	Discard, Deliver Unconditionally, or Deliver Conditionally. If Deliver Conditionally is used, then the conditions must be specified.
Layer 2 Control Protocols Processing	A list of Layer 2 Control Protocols labeled Tunnel or Discard.

Ethernet Virtual Connection

An Ethernet Virtual Connection (EVC) is a logical relationship between both user and network interfaces in a provider-based Ethernet environment. It establishes communication relationship between UNI- and UNI-N devices. There are three types of Metro Ethernet services provided by the EVC:

- Point-to-point
- Multipoint-to-multipoint
- Rooted-multipoint

Point-to-Point EVC

Point-to-point supports communication between exactly two UNIs. That is, an ingress service frame mapped to the EVC at one UNI must map to an ingress service frame at another UNI. The rules under which a service frame delivers to the destination UNI are specific to the particular service definition.

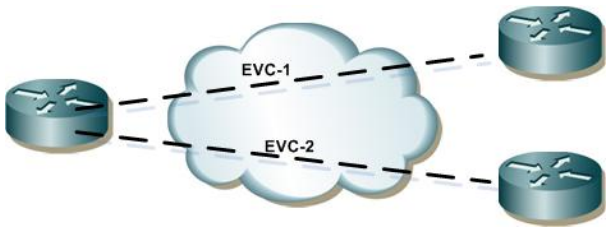


Figure 13-4: Point-to-point ME Services

Multipoint-to-Multipoint EVC

Multipoint-to-multipoint supports any-to-any communication between two or more UNIs. In a multipoint-to-multipoint EVC, a single broadcast or multicast ingress service frame (as determined from the destination MAC address) at one UNI would be replicated in the Metro Ethernet Network and a single copy would be delivered to each of the other UNIs in the EVC.

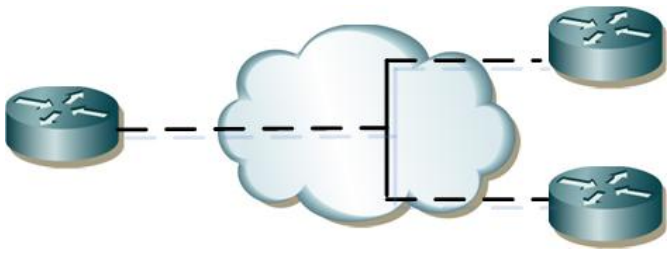


Figure 13-5: Multipoint-to-Multipoint ME Services

Rooted-Multipoint EVC

In a rooted-multipoint EVC, one or more of the UNIs must be designated as root and each of the other UNIs must be designated as a leaf. If root, the UNI can send service frames to all other points in the EVC; if leaf, the UNI can send and receive service frames to and from root only.

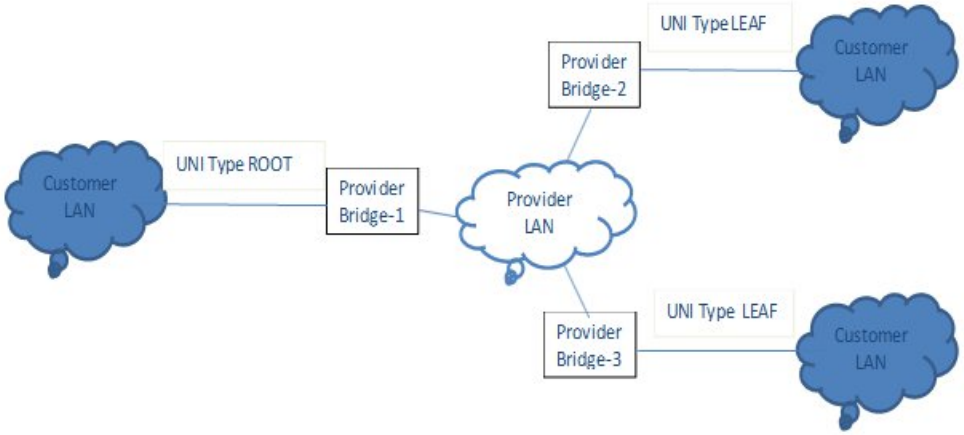


Figure 13-6: Rooted-Multipoint ME Services

Bandwidth Profile

A bandwidth profile is a method of characterizing service frames for the purpose of rate enforcement or policing. There are two types of Bandwidth Profile:

- An Ingress Bandwidth profile helps regulate the amount of ingress traffic at a particular UNI
- An Egress Bandwidth profile helps regulate the amount of egress traffic at a particular UNI

A Bandwidth Profile is a characterization of the lengths and arrival times for service frames at a reference point. For the Ingress Bandwidth profile, this reference point is the Ingress UNI. For the Egress Bandwidth Profile, this reference point is the Egress UNI.

Note: The bandwidth profile can be tested only on hardware with that capability. The mls qos implementation can be re-used for the same. See the *Quality of Service Command Reference* and the *Quality of Service Configruation Guide* for details about bandwidth profiling.

Bandwidth Profile Types

The bandwidth profiling can be one of these types:

- Ingress/Egress bandwidth profiling per UNI
- Ingress/Egress bandwidth profiling per EVC
- Ingress/Egress bandwidth profiling per class of service (CoS)

Ingress/Egress Bandwidth Profile per UNI

A single Ingress Bandwidth profile applies to all ingress service frames at UNI. The Ingress Bandwidth profile per Ingress UNI manages bandwidth non-discriminatively for all EVCs at UNI. This means that some EVCs may get more bandwidth than others. A single Egress Bandwidth profile applies to all egress service frames at the egress UNI.

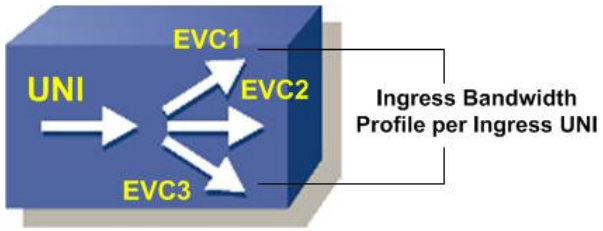


Figure 13-7: Ingress Bandwidth Profile per Ingress UNI

Egress Bandwidth Profile per EVC

If a user defines a single Egress Bandwidth Profile for an EVC at the egress UNI, then it must apply to the egress service frames mapped to the EVC.

Ingress Bandwidth Profile per EVC

A single Ingress Bandwidth Profile applies to all ingress service frames for an instance of an EVC at UNI. If a UNI has three Ethernet Virtual Connections, there could be three Ingress Bandwidth profiles, or one Ingress Bandwidth profile for each EVC.

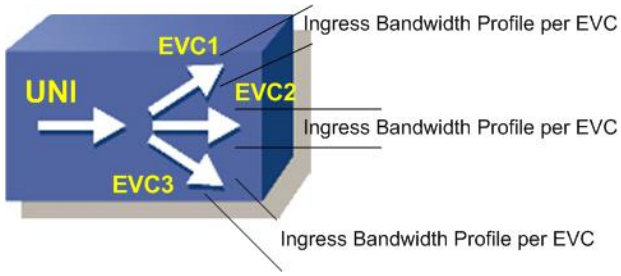


Figure 13-8: Ingress Bandwidth Profile per EVC

Ingress/Egress Bandwidth Profiling per CoS Identifier

A single Ingress Bandwidth Profile applies to all ingress service frames with a specific Class of Service (CoS) Identifier. For example, there can be three CoS Identifiers within EVC, each with a separate Ingress Bandwidth Profile.

Note: Configure UNI so that only a single Ingress Bandwidth Profile applies to any given ingress service frame. The following two points apply for an Egress Bandwidth Profile, as well. For example:

- If there is a UNI Ingress Bandwidth Profile, then there cannot be any other Ingress Bandwidth Profiles at that UNI.
- If there is an EVC Ingress Bandwidth Profile on an EVC, then there cannot be any per Class of Service Ingress Bandwidth Profiles for instances of CoS on that EVC.

Bandwidth Profile Parameters

The following are the bandwidth profile parameters for UNI:

- Committed Information Rate (CIR) expressed as bits per second. CIR must be less than or greater to zero.
- Committed Burst Size (CBS) expressed as bytes. When CIR is less than zero, then the CBS must be greater than or equal to the largest Maximum Transmission Unit (MTU) size among all of the EVCs wherever the Bandwidth Profile applies.
- Excess Information Rate (EIR) expressed as bits per second. EIR MUST be less than or greater to zero.
- Excess Burst Size (EBS) expressed as bytes. When EIR is less than zero, then EBS must be greater than or equal to the largest MTU size among all EVCs where the Bandwidth Profile applies.
- Coupling Flag (CF) has only one of two possible values: 0 or 1 (that is, disabled or enabled).
- Color Mode (CM) has only one of two possible values: color-blind or color-aware.

CIR and CBS

The CIR and EIR must be less than or equal to the speed of UNI. Service frames whose average rate is greater than the CIR are not CIR-compliant and are either colored yellow (if EIR is non-zero) or are discarded (if EIR = 0).

EIR and EBS

EBS is the maximum number of bytes allowed for incoming service frames to be EIR-compliant. EIR-compliant service frames are colored yellow. Service frames whose average rate is greater than the EIR are not EIR-compliant and are discarded. These frames are colored red.

Color Mode

A color-blind UNI is one where the bandwidth profile algorithm ignores any color indication that the subscriber may have marked in their service frames.

Data Structures

The functions in this chapter refer to the data structures described in this section.

nsm_vlan

This structure in `nsm/L2/nsm_vlan.h` defines information about a VLAN.

Member name	Description
`vlan_name`	VLAN name
`bridge`	Back pointer to bridge
`vid`	VLAN identifier
`type`	VLAN type
`ifp`	Pointer to interface

Member name	Description
pvlan_configured	PVLAN configuration
pvlan_type	PVLAN type
pvlan_info	PVLAN information
evc_id	SVLAN identifier
cvlanMemberBmp	CVLAN member bitmap
preserve_ce_cos	Preserve class of service
max_uni	MAX UNI per EVC: MEF 10.1, Table 16
dscp_cos	DCSP COS
pcp_cos	PCP COS
pcp_dscp_flag	PCP DSCP flag
vlan_state	VLAN state
mtu_val	VLAN MTU value
port_list	Port list for the VLAN
forbidden_port_list	Forbidden port list for the VLAN
vlan_filter	VLAN filter
static_fdb_list	Static FDB list
nsm_vlan_cdr_ref	VLAN CDR reference
port_cdr_ref_list	Port CDR reference list
unicast_egress_ports	The set of ports to which frames received from a specific port and destined for a specific unicast address must be forwarded
unicast_forbidden_ports	The set of ports to which frames received from a specific port and destined for a specific unicast address must not be forwarded
multicast_egress_ports	The set of ports to which frames received from a specific port and destined for a specific multicast or broadcast MAC address must be forwarded
multicast_forbidden_ports	The set of ports to which frames received from a specific port and destined for a specific multicast or broadcast MAC address must not be forwarded
priority	VLAN priority
create_time	VLAN create time
primary_vid	Whether this VID is configured as a primary VID

```
struct nsm_vlan
{
```

```
#define NSM_VLAN_NAME_MAX   32
  /* VLAN name. */
  char                      vlan_name[NSM_VLAN_NAME_MAX + 1];

  /* Back pointer to bridge */
  struct nsm_bridge         *bridge;

  /* VID */
  nsm_vid_t                 vid;

  /* Type. */
  u_int16_t type;
#define NSM_VLAN_STATIC     (1 << 0)
#define NSM_VLAN_DYNAMIC    (1 << 1)
#define NSM_VLAN_CVLAN      (1 << 2)
#define NSM_VLAN_SVLAN      (1 << 3)
#define NSM_SVLAN_P2P       (1 << 4)
#define NSM_SVLAN_M2M       (1 << 5)
#ifdef HAVE_B_BEB
#define NSM_VLAN_BVLAN      (1 << 6)
#define NSM_BVLAN_P2P       (1 << 7)
#define NSM_BVLAN_M2M       (1 << 8)
#endif /* HAVE_I_BEB */
#ifdef HAVE_PBB_TE
#define NSM_VLAN_TEVLAN     (1 << 9)
#endif
#define NSM_VLAN_AUTO       (1 << 10)

  struct interface *ifp;

#ifdef HAVE_PVLAN
  int pvlan_configured;

  int pvlan_type;

  /*struct list *secondary_vlan_list;*/
  struct nsm_pvlan_info pvlan_info;
#endif /* HAVE_PVLAN */

  u_int8_t *evc_id; ///< EVC ID is the id for svlan as per MEF 10.1

#ifdef HAVE_PROVIDER_BRIDGE

  struct nsm_vlan_bmp *cvlanMemberBmp;
  u_int8_t preserve_ce_cos;

  u_int16_t max_uni; ///<MAX UNI Per EVC as per MEF 10.1, Table 16

#define NSM_UNI_EVC_COS_PCP 8
#define NSM_UNI_EVC_COS_DSCP 64
```

```
  u_int8_t dscp_cos[NSM_UNI_EVC_COS_DSCP];
  u_int8_t pcp_cos[NSM_UNI_EVC_COS_PCP];

  u_int8_t pcp_dscp_flag;
#endif /* HAVE_PROVIDER_BRIDGE */

  /* VLAN state. */
  enum nsm_vlan_state       vlan_state;

  /* VLAN Mtu. */

  u_int32_t mtu_val;

  /* VLAN port list. */
  struct list               *port_list;

  /* Forbidden port list for the vlan */
  struct list               *forbidden_port_list;

#ifdef HAVE_L2LERN
  struct nsm_vlan_access_list   *vlan_filter;
#endif /* HAVE_L2LERN */

 /* Static FDB List */
  struct ptree              *static_fdb_list;

  int instance;
#ifdef HAVE_HA
  HA_CDR_REF                nsm_vlan_cdr_ref;
  struct list               *port_cdr_ref_list;
#endif /* HAVE_HA */

#ifdef HAVE_PBB_TE
  /* The set of ports to which frames received from a specific port
  and destined for a specific unicast address must be forwarded */
  struct list *unicast_egress_ports;
  /* The set of ports to which frames received from a specific port
  and destined for a specific unicast address must not be forwarded */
  struct list *unicast_forbidden_ports;
  /* The set of ports to which frames received from a specific port
  and destined for a specific Multicast or Broadcast MAC address must be forwarded */
  struct list *multicast_egress_ports;
  /* The set of ports to which frames received from a specific port
  and destined for a specific Multicast or Broadcast MAC address
  must not be forwarded*/
  struct list *multicast_forbidden_ports;
#endif /* HAVE_PBB_TE */

#ifdef HAVE_QOS
```



```
#define NSM_VLAN_PRIORITY_CONF      (1 << 5)
#define NSM_VLAN_PRIORITY_MASK      (0x7)
#define NSM_VLAN_PRIORITY_NONE      (0x0)

  u_int8_t priority;
#endif /* HAVE_QOS */

#ifdef HAVE_SNMP
  pal_time_t create_time;
#endif /* HAVE_SNMP */
#ifdef HAVE_G8032
  /* Whether this vid is configured as a primary vid
   * for any of the g8032 rings
   */
  u_int8_t primary_vid;
#endif /* HAVE_G8032 */
};
```

nsm_bridge

This structure is in `nsm/L2/nsm_bridge.h`

```
struct nsm_bridge
{
  ....
  ....
   struct avl_tree *pro_edge_swctx_table;
   struct list *cvlan_reg_tab_list;
+  struct list *oep_map_tab_list;
   struct list *vlan_listener_list;
 #ifdef HAVE_B_BEB
   struct avl_tree *bvlan_table;
  ....
  ....
}
```

nsm_ovc_info

This structure is in `nsm/L2/nsm_pro_vlan.h`

```
+struct nsm_ovc_info
+{
+  u_int8_t *ovc_id;
+  u_int8_t *oep_id;
+  u_int8_t preserve_vlan;
+  u_int16_t num_ovcs;
+#define NSM_CVLAN_COS_PRESERVE  (1 << 0)
+#define NSM_CVLAN_ID_PRESERVE   (1 << 1)
+#define NSM_SVLAN_COS_PRESERVE  (1 << 2)
+#define NSM_SVLAN_ID_PRESERVE   (1 << 3)
+};
+
```

```
+struct nsm_oep_map_trans_key
+{
+  u_int16_t vid;
+  u_int16_t ovc_vid;
+};
+
+struct nsm_oep_info
+{
+  struct nsm_vlan_bmp ennisvlanBmp;
+  struct nsm_ovc_info *ovc_info;
+  struct list *port_list;
+  u_int16_t svid;
+  /* Required to add the map on a UNI */
+  u_int16_t ovc_type;
+};
+
+struct nsm_oep_map_tab
+{
+  u_int8_t type;
+#define NSM_OEP_MAP_TYPE_INVALID  (1 << 0)
+#define NSM_OEP_MAP_TYPE_UNI      (1 << 1)
+#define NSM_OEP_MAP_TYPE_ENNI     (1 << 2)
+  struct nsm_bridge *bridge;
+  struct list *port_list;
+  struct avl_tree *oep_map;
+  struct avl_tree *oep_tree;
+#define NSM_OEP_MAP_TAB_NAME_MAX 16
+  char name [NSM_OEP_MAP_TAB_NAME_MAX + 1];
+
+};
+
```

nsm_ovc_info

This structure is in `nsm/L2/nsm_pro_vlan.h`

Command API

The functions in this section are called by the commands in the *Carrier Ethernet Command Reference*.

Function	Description
nsm_bridge_api_port_proto_process	Configures the protocol handling on a customer edge/customer network port
nsm_bridge_uni_set_name	Deletes the name of an UNI
nsm_bridge_uni_type_detect	Enables or disables the UNI type mode for an interface
nsm_bridge_uni_unset_name	Removes the name of an UNI

Include Files

Except where noted otherwise, you need to include `nsm/L2/nsm_pro_vlan.h` to call the functions in this section.

nsm_bridge_api_port_proto_process

The function configures the protocol handling on a customer edge/customer network port.

This function is called by the `l2protocol` command.

Syntax

```
int
nsm_bridge_api_port_proto_process (struct interface *ifp,
                                   enum hal_l2_proto proto,
                                   enum hal_l2_proto_process process,
                                   bool_t iterate_members,
                                   u_int16_t svid, u_int8_t cos_id)
```

Input Parameters

`ifp` Interface pointer

`proto` Protocol; one of these constants from the `hal_l2_proto` enum in `hal/L2/hal_bridge.h`:

- `HAL_PROTO_STP` Spanning-Tree Protocol
- `HAL_PROTO_RSTP` Rapid Spanning-Tree Protocol
- `HAL_PROTO_MSTP` Multiple Spanning-Tree Protocol
- `HAL_PROTO_TRILL` TRILL
- `HAL_PROTO_GMRP` GARP Multicast Address Registration Protocol
- `HAL_PROTO_MMRP` Multiple Multicast Routing Protocol
- `HAL_PROTO_GVRP` GARP VLAN Registration Protocol
- `HAL_PROTO_MVRP` Multiple VLAN Registration Protocol
- `HAL_PROTO_LACP` Link Aggregation Control Protocol
- `HAL_PROTO_DOT1X` IEEE 802.1x port authentication

`process` Action to take for the protocol; one of these constants from the `hal_l2_proto_process` enum in `hal/L2/hal_bridge.h`:

HAL_L2_PROTO_TUNNEL

Tunnel the specified protocol on the interface or service VLAN

HAL_L2_PROTO_PEER

Peer the specified protocol on the interface or service VLAN. This setting cannot be used with the GVRP or MVRP protocols.

HAL_L2_PROTO_DISCARD

Discard the specified protocol packets on the interface or service VLAN

`iterate_members`

Whether to iterate members (LACP only):

`PAL_TRUE`	Iterate members
`PAL_FALSE`	Do not iterate members

`svid`	SVLAN identifier
`cos_id`	Class of Service (CoS) identifier

Output Parameters

None

Return Values

NSM_BRIDGE_ERR_NOTFOUND when this operation is not appropriate for the interface configuration

NSM_VLAN_ERR_VLAN_NOT_FOUND when a VLAN is not found

NSM_BRIDGE_ERR_INVALID_MODE when this operation is not appropriate for the interface configuration

NSM_BRIDGE_ERR_INVALID_PROTO when the protocol is invalid

RESULT_OK when the function succeeds

nsm_bridge_uni_set_name

The function sets the name of an UNI.

This function is called by the `ethernet uni id` command.

Syntax

```
s_int32_t
nsm_bridge_uni_set_name (struct interface *ifp,char *uni_name,
                         char *uni_type,u_int16_t svlan_id,char * evc_id)
```

Input Parameters

`ifp`	Interface pointer
`uni_name`	The name of the UNI
`uni_type`	UNI type:
`"root"`	The UNI can send service frames to all other points in the EVC
`"leaf"`	The UNI can send and receive and service frames to and from `root` only
`svlan_id`	SVLAN identifier <2-4094>
`evc_id`	EVC identifier

Output Parameters

None

Return Values

NSM_ERR_UNI_EXISTS when the UNI already has this name

NSM_ERR_UNI_DUPLICATE_ENTRY when a UNI on another interface already has this name

NSM_L2_ERR_INVALID_ARG when this operation is not appropriate for the interface configuration

NSM_PRO_VLAN_ERR_INVALID_MODE when this operation is not appropriate for the interface configuration

NSM_ERR_INVALID_SVLAN_OR_EVC_ID when the SVLAN is NULL

NSM_ERR_INVALID_EVC_UNI_TYPE when the UNI type is leaf, but the SVLAN is rooted multipoint

NSM_ERR_UNIPAIR_LIST_CREATE_FAIL when the UNI type/SVLAN pair cannot be created

NSM_ERR_UNIPAIR_LIST_ADD_FAIL when the UNI type/SVLAN pair cannot be added

NSM_PRO_VLAN_ERR_HAL_ERR when the UNI cannot be added to HAL

NSM_L2_ERR_NONE when the function succeeds

nsm_bridge_uni_type_detect

The function enables or disables the UNI type mode for an interface.

Syntax

```
s_int32_t
nsm_bridge_uni_type_detect (u_int8_t *br_name, u_int8_t uni_type_mode,
                            struct nsm_bridge_master *master)
```

Input Parameters

`br_name`	The bridge name
`uni_type_mode`	Whether to enable or disable
`master`	NSM bridge master

Output Parameters

None

Return Values

NSM_BRIDGE_ERR_NOTFOUND when the bridge is not found

NSM_BRIDGE_ERR_INVALID_ARG when the bridge is not configured as provider edge

RESULT_OK when the function succeeds

nsm_bridge_uni_unset_name

The function deletes the name of an UNI.

This function is called by the `no ethernet uni id` command.

Syntax

```
s_int32_t
nsm_bridge_uni_unset_name (struct interface *ifp,char * uni_name,
                           s_int16_t svlan_id,char * evc_id)
```

Input Parameters

`ifp`	Interface pointer
`uni_name`	The name of the UNI
`svlan_id`	SVLAN identifier <2-4094>
`evc_id`	EVC identifier

Output Parameters

None

Return Values

NSM_L2_ERR_INVALID_ARG when this operation is not appropriate for the interface configuration

NSM_PRO_VLAN_ERR_INVALID_MODE when this operation is not appropriate for the interface configuration

NSM_NO_UNI_EXIST when a UNI name is not set

NSM_INVALID_UNI_ID when the given UNI name does not match the configured UNI name

RESULT_OK when the function succeeds

nsm_svlan_set_evc_id

This function sets the identifier of an EVC.

Syntax

```
s_int32_t
nsm_svlan_set_evc_id (struct nsm_bridge_master *master,
                      u_int8_t *bridge_name, u_int16_t svid, u_int8_t *evc_id)
```

Input Parameters

`master`	NSM pointer
`bridge_name`	Name of the bridge
`svid`	SVLAN identifier
`evc_id`	EVC identifier; must be unique across all EVCs in the MEN

Output Parameters

None

Return Values

NSM_VLAN_ERR_BRIDGE_NOT_FOUND when the bridge is not found

NSM_VLAN_ERR_BRIDGE_NOT_VLAN_AWARE when the bridge is not VLAN aware

NSM_VLAN_ERR_VLAN_NOT_FOUND when the VLAN is not found

NSM_VLAN_ERR_EVC_ID_SET If the EVC ID is already set

RESULT_OK when the function succeeds

nsm_svlan_set_max_uni

This function sets the maximum number of UNIs allowed in the UNI list service attribute. This must be two if EVC type is point-to-point.

API

```
s_int32_t
nsm_svlan_set_max_uni (struct nsm_bridge_master *master,
                       u_int8_t *bridge_name, u_int16_t svid, u_int8_t *evc_id,
                       u_int16_t max_uni)
```

Input Parameters

`master`	Bridge master
`bridge_name`	Bridge name
`SVLAN`	SVLAN identifier
`evc_id`	EVC identifier
`max_uni`	Maximum number of UNIs

Output Parameters

None

Return Values

NSM_VLAN_ERR_BRIDGE_NOT_FOUND when the bridge is not found

NSM_VLAN_ERR_NOT_EDGE_BRIDGE when the bridge is not configured as an edge bridge

NSM_VLAN_ERR_BRIDGE_NOT_VLAN_AWARE when the bridge is not VLAN aware

NSM_VLAN_ERR_VLAN_NOT_FOUND when the VLAN is not found

NSM_VLAN_ERR_MAX_UNI_PT_TO_PT when the VLAN is point-to-point and `max_uni` is not two

NSM_VLAN_ERR_MAX_UNI_M2M when the SVLAN is multipoint-to-multipoint and `max_uni` is less than 2

RESULT_OK when the function succeeds

nsm_svlan_set_cvlanid_preservation

This function to set vlanid preservation.

Syntax

```
s_int32_t
nsm_svlan_set_cvlanid_preservation (struct nsm_bridge_master *master,
                                    char *bridge_name, u_int16_t vid,
                                    u_int8_t preserve_ce_vlanid,
                                    bool_t *warning_flag )
```

Input Parameters

`master`	NSM bridge master

`bridge_name` Name of the bridge

Output Parameters

None

Return Values

NSM_L2_ERR_INVALID_ARG when the port is not configured for switching

NSM_PRO_VLAN_ERR_INVALID_MODE when the port is not configured as a customer edge

NSM_ERR_BW_PROFILE_NOT_CONFIGURED when the bandwidth profile is not configured

NSM_ERR_BW_PROFILE_PER_EVC_CONFIGURED when the bandwidth profile per EVC is configured

RESULT_OK when the function succeeds

CHAPTER 14 External Network Network Interface

ZebOS-XP ENNI provide a reference point representing the boundary between two Operator MENs that are operated as separate administrative domains. It offers a reference point for Ethernet service delivery by establishing a standards-based connection through a Metro Ethernet Network (MEN).

Overview

An ENNI can be implemented with one or more physical links. However, when there is no protection mechanism among multiple links connecting two Operator MENs, each link represents a distinct ENNI. Figure 14-1 displays a simple diagram of an ENNI operation.

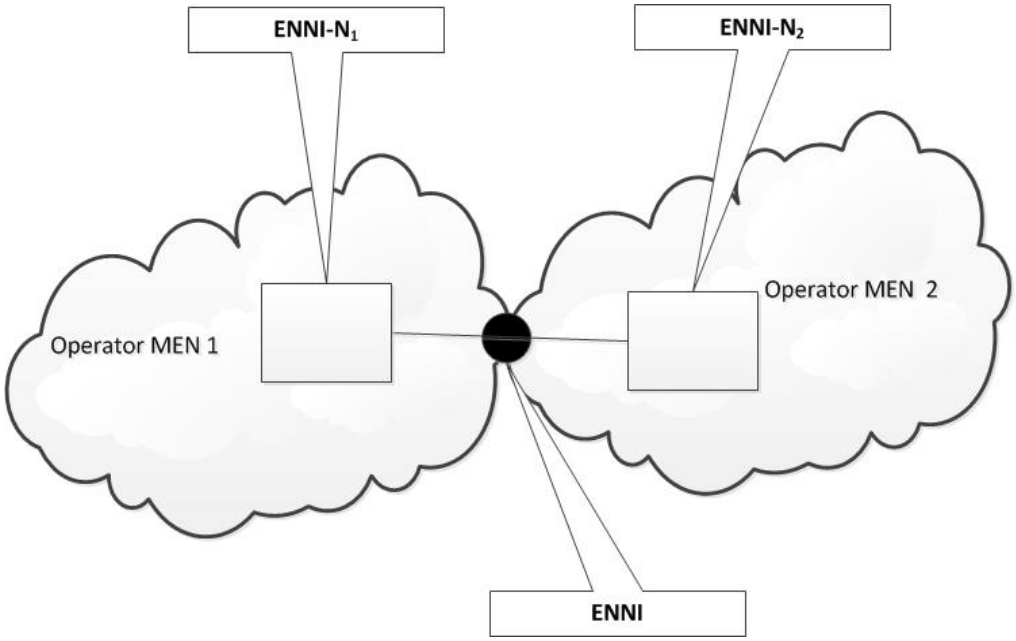


Figure 14-1: UNI Operation

Feature

The following are some of the features of ENNI.

- Full implementation and support for the following:
  - MEF-6.1 and MEF-10.2
  - MEF 6.1.1 and MEF 10.2.1
  - MEF 26.1 and MEF 33
- Provides support for point-to-point, multipoint-to-multipoint and rooted multipoint configurations in a service provider network for both port-based and vlan-based services.

Provides support for point-to-point E-Access configurations in a service provider network for both port-based and vlan-based services.

Operator Virtual Connection

An Operator Virtual Connection (OVC) is the building block for constructing an EVC spanning multiple Operator MENs. An OVC can informally be thought of as an association of “External Interfaces” within the same Operator MEN. This association constrains the delivery of frames in the sense that an egress Service Frame or ENNI Frame mapped to a given OVC can only be the result of an ingress Service Frame or ENNI Frame mapped to the given OVC.

In the case of an ENNI, an egress ENNI Frame with identical MAC and payload information can result from an ingress ENNI Frame at the same interface.

Note: This behavior is not allowed at a UNI as specified in MEF 10.2 [5].

To describe this behavior, the OVC End Point is used which allows multiple, mutually exclusive ways that an ENNI Frame can be mapped to a single OVC at an ENNI. Section 7.2 defines the OVC as an association of OVC End Points. In turn each OVC End Point is associated with either a UNI or an ENNI. For the scope of this document, at least one OVC End Point associated by an OVC is at an ENNI.

Types of Ethernet services

- E-line Services - The Ethernet Private Line (EPL) and Ethernet Virtual Private Line (EVPL) services as defined in MEF 6.1 can be used to create a broad range of point-to-point services.
- E-LAN services - The Ethernet Private-LAN (EP-LAN) and Ethernet Virtual Private-LAN (EVP-LAN) services as defined in MEF 6.1 can be used to create a broad range of multipoint-to-multipoint services.
- E-Tree Services The EP-Tree and EVP-Tree services as defined in MEF 6.1 can be used to create a broad range of rooted-multipoint services.
- E-Access Services - An Access EPL service MUST use a Point-to-Point OVC that associates one OVC End Point at a UNI and one OVC End Point at an ENNI.

Data Structures

The functions in this chapter refer to the data structures described in this section.

nsm_ingress_egress_svid

This structure in `nsm/L2/nsm_vlan.h` creates the parameters while entering into service instance mode.

```
/*Create the paramteres while entering into service instance mode */
struct nsm_ingress_egress_ovc_svid
{
  struct nsm_enni_ovc_bw_profile *ingress;
  struct nsm_enni_ovc_bw_profile *egress;
};
 /* The structure maintated per OVC to store the BW Profile for the OVC */
struct nsm_enni_ovc_bw_profile
{
  u_int16_t svlan; ///< svlan to which this enni ovc bw profile is applied
   u_int8_t *ovc_id; ///< OVC-ID
   u_int16_t instance_id; ///< INSTANCE ID
   struct interface *ifp; ///< backpointer to the interface
  u_int8_t cos_id;  ///< added to index based on cos value
  u_int8_t CoS_ID;   ////< cos id
```



```
  u_int8_t ingress_bw_profile_type; ///< Per svlan or Per CoS
  u_int8_t egress_bw_profile_type;  ///< Per svlan or Per CoS
  u_int8_t bw_profile_type;  ///< ingree or egress
  struct nsm_bw_profile_parameters *bw_profile_param;  ////<pointer to bw profile
paramteres
 #define NSM_EVC_BW_PROFILE_PER_EVC     1
#define NSM_EVC_COS_BW_PROFILE         2
   struct nsm_bw_profile_parameters *ingress_ovc_bw_profile;
   struct nsm_bw_profile_parameters *egress_ovc_bw_profile;
 #define NSM_MAX_COS_PER_EVC 8
  u_int8_t num_of_ingress_cos_id [NSM_MAX_COS_PER_EVC]; ///< Array maintained to
    ///<assign a COS ID from this array for the cos values set for bw profiling
   u_int8_t ingress_CoS_id [NSM_MAX_COS_PER_EVC];  ///< Array used to maintain
    ///<  the the cos id generated for each cos value set
   u_int8_t num_of_egress_cos_id [NSM_MAX_COS_PER_EVC];   ///< Array maintained
    ///< to assign COS ID from this array for the cos values set for profiling
   u_int8_t egress_CoS_id [NSM_MAX_COS_PER_EVC];   ///< Array used to maintain
    ///< the the cos id generated for each cos value set
   #define NSM_BW_PROFILE_PER_EVC 1
 struct nsm_bw_profile_parameters
    *ingress_ovc_cos_bw_profile [NSM_MAX_COS_PER_EVC]; ///< Array of bw profile
                                      ///<   parameters for each cos value set
   struct nsm_bw_profile_parameters
    *egress_ovc_cos_bw_profile [NSM_MAX_COS_PER_EVC]; ///< Array of bw profile
                                     ///<   parameters for each cos value set
#define NSM_BW_PROFILE_PER_EVC 1
#define NSM_BW_PROFILE_PER_COS 2
#define NSM_MAX_COS_PER_EVC 8
 };
```

nsm_ovc_band_width_profile

This structure is in `nsm/L2/nsm_vlan.h`.

```
struct nsm_ovc_band_width_profile
{
  /* Back Pointer to ifp */
  struct nsm_if *zif;

  struct nsm_bw_profile_parameters *ingress_enni_bw_profile; ///< Pointer
            ///<  maintatined to store the bw profile parameters per ENNI

  struct nsm_bw_profile_parameters *egress_enni_bw_profile;  ///< Pointer
            ///<  maintatined to store the bw profile parameters per ENNI

  struct avl_tree *enni_ovc_bw_profile_list;  ///< List maintained on each interface
            ///< for the ovcs associated to the interface.

  u_int8_t ingress_bw_profile_type; ///< Per EVC or Per ENNI

  u_int8_t egress_bw_profile_type; ///< Per EVC or Per ENNI
```

```
    #define NSM_BW_PROFILE_PER_EVC     1
    #define NSM_BW_PROFILE_PER_UNI     2

  u_int8_t ingress_bw_profile_status; ///<BW_PROFILE Configured or not

  u_int8_t egress_bw_profile_status; ///<BW_PROFILE Configured or not

  #define BW_PROFILE_CONFIGURED      1
  #define BW_PROFILE_NOT_CONFIGURED 0

  struct nsm_bridge *bridge; ///< Backpointer to the bridge
};
```

Command API

The functions in this section are called by the commands in the *Carrier Ethernet Command Reference*.

Include Files

Except where noted otherwise, you need to include `nsm/L2/nsm_pro_vlan.h` to call the functions in this section.

nsm_enni_bw_profiling

This function configures a bandwidth profile for a ENNI.

Syntax

```
s_int32_t
nsm_enni_bw_profiling (struct interface *ifp, u_int64_t cir, u_int32_t cbs,
    u_int64_t eir, u_int32_t ebs, u_int8_t coupling_flag, u_int8_t color_mode,
    u_int8_t bw_profile_type, u_int8_t bw_profile_parameter)
```

Input Parameters

`ifp`	Interface pointer
`cir`	Committed information rate
`cbs`	Committed burst rate
`eir`	Excess information rate
`ebs`	Excess burst rate
`coupling_flag`	Whether the coupling flag is enabled or disabled:
`PAL_TRUE`	Enabled
`PAL_FALSE`	Disabled
`color-mode`	Whether color-aware or color-blind
`PAL_TRUE`	Color-aware
`PAL_FALSE`	Color-blind

`bw_profile_type`

Ingress or egress; one of these constants from `nsm/L2/nsm_vlan.h`:

`NSM_INGRESS_POLICING`

`NSM_EGRESS_SHAPING`

`bw_profile_parameter`

Bandwidth profile parameter to set; one of these constants from `nsm/L2/nsm_vlan.h`:

`NSM_BW_PARAMETER_CIR`

`NSM_BW_PARAMETER_CBS`

`NSM_BW_PARAMETER_EIR`

`NSM_BW_PARAMETER_EBS`

`NSM_BW_PARAMETER_COUPLING_FLAG`

`NSM_BW_PARAMETER_COLOR_MODE`

Output Parameters

None

Return Values

NSM_L2_ERR_INVALID_ARG when the port is not configured for switching

NSM_PRO_VLAN_ERR_INVALID_MODE when the port is not configured as a customer edge

NSM_ERR_BW_PROFILE_ACTIVE when the bandwidth profile is active

NSM_ERR_BW_PROFILE_PER_OVC_CONFIGURED when the bandwidth profile per OVC is configured

RESULT_OK when the function succeeds

nsm_delete_enni_bw_profiling

This function deletes a bandwidth profile.

Syntax

```
s_int32_t
nsm_delete_enni_bw_profiling (struct interface *ifp, u_int8_t bw_profile_type)
```

Input Parameters

`ifp` Interface pointer

`bw_profile_type`

Ingress or egress; one of these constants from `nsm/L2/nsm_vlan.h`:

`NSM_INGRESS_POLICING`

`NSM_EGRESS_SHAPING`

Output Parameters

None

Return Values

NSM_ERR_NOT_FOUND when the interface information is not found

NSM_L2_ERR_INVALID_ARG when the port is not configured for switching

NSM_PRO_VLAN_ERR_INVALID_MODE when the port is not configured as a customer edge

NSM_ERR_BW_PROFILE_NOT_CONFIGURED when the bandwidth profile is not configured

NSM_ERR_BW_PROFILE_ACTIVE when the bandwidth profile is active

RESULT_OK when the function succeeds

nsm_set_enni_bw_profiling

This function activates or deactivates a bandwidth profile.

Syntax

```
s_int32_t
nsm_set_enni_bw_profiling (struct interface *ifp, u_int8_t bw_profile_type,
    u_int8_t bw_profile_status)
```

Input Parameters

`ifp` Interface pointer

`bw_profile_type`

Ingress or egress; one of these constants from `nsm/L2/nsm_vlan.h`:

`NSM_INGRESS_POLICING`

`NSM_EGRESS_SHAPING`

`bw_profile_status`

Whether to activate or deactivate the bandwidth profile

Output Parameters

None

Return Values

NSM_ERR_NOT_FOUND when the interface information is not found

NSM_L2_ERR_INVALID_ARG when the port is not configured for switching

NSM_PRO_VLAN_ERR_INVALID_MODE when the port is not configured as a customer edge

NSM_ERR_BW_PROFILE_NOT_CONFIGURED when the bandwidth profile is not configured

NSM_ERR_BW_PROFILE_PER_OVC_CONFIGURED when the bandwidth profile per OVC is configured

NSM_ERR_BW_PROFILE_CBS_NOT_CONFIGURED when the CBS value is less than MTU

NSM_ERR_BW_PROFILE_EBS_NOT_CONFIGURED when the EBS value is less than MTU

NSM_ERR_BW_PROFILE_EIR_NOT_CONFIGURED when the committed information rate is greater than the excess information rate

NSM_ERR_BW_PROFILE_NOT_ACTIVE when the bandwidth profile is not active

RESULT_OK when the function succeeds

nsm_enni_ovc_set_service_instance

This function is used to set the service instance id for svlan/ovc-id.

Syntax

```
s_int32_t
nsm_enni_ovc_set_service_instance (struct interface *ifp, u_int16_t instance_id,
    u_int8_t *ovc_id, struct nsm_ingress_egress_svid
    **ret_nsm_enni_ovc_bw_profile)
```

Input Parameters

`ifp`	Interface pointer
`instance_id`	Ingress or egress; one of these constants from `nsm/L2/nsm_vlan.h`:
`ovc_id`	The ovc-id for which the service instance mode need to be entered.
`ret_nsm_enni_ovc_bw_profile`	Pointer which is updated with the OVC_bw_profile node

Output Parameters

None

Return Values

RESULT_OK when the function succeeds

nsm_unset_enni_ovc_set_service_instance

This function deletes a service instance.

Syntax

```
s_int32_t
nsm_unset_enni_ovc_set_service_instance (struct interface *ifp, u_int16_t instance_id,
u_int8_t *ovc_id)Input Parameters
```

`ifp`	Interface pointer
`instance_id`	Instance identifier
`ovc_id`	OVC identifier

Output Parameters

None

Return Values

NSM_L2_ERR_INVALID_ARG when the port is not configured for switching

NSM_PRO_VLAN_ERR_INVALID_MODE when the port is not configured as a customer edge

NSM_ERR_BW_PROFILE_NOT_CONFIGURED when the bandwidth profile is not configured

NSM_ERR_BW_PROFILE_PER_OVC_CONFIGURED when the bandwidth profile per OVC is configured

RESULT_OK when the function succeeds

nsm_delete_ovc_bw_profiling

This function deletes a bandwidth profile.

Syntax

```
s_int32_t
nsm_delete_ovc_bw_profiling (struct interface *ifp, struct nsm_ingress_egress_ovc_svid
*dual,  u_int8_t bw_profile_type)
```

Input Parameters

`ifp` Interface pointer

`dual` Pointer to `nsm_ingress_egress_svid`

`bw_profile_type`

Ingress or egress; one of these constants from `nsm/L2/nsm_vlan.h`:

`NSM_INGRESS_POLICING`

`NSM_EGRESS_SHAPING`

Output Parameters

None

Return Values

NSM_ERR_NOT_FOUND when the interface information is not found

NSM_L2_ERR_INVALID_ARG when the port is not designated as an L2 port

NSM_PRO_VLAN_ERR_INVALID_MODE when the port is not configured as a customer edge

NSM_ERR_BW_PROFILE_NOT_CONFIGURED when the bandwidth profile is not configured

NSM_ERR_BW_PROFILE_ACTIVE when the bandwidth profile is activated

RESULT_OK when the function succeeds

nsm_set_ovc_bw_profiling

This function activates or deactivates a bandwidth profile.

Syntax

```
s_int32_t
nsm_set_ovc_bw_profiling (struct interface *ifp,
    struct nsm_ingress_egress_svid *dual, u_int8_t bw_profile_type,
    u_int8_t bw_profile_status)
```

Input Parameters

`ifp` Interface pointer

`dual` Pointer to `nsm_ingress_egress_svid`

`bw_profile_type`

Ingress or egress; one of these constants from `nsm/L2/nsm_vlan.h`:

`NSM_INGRESS_POLICING`

NSM_EGRESS_SHAPING

bw_profile_status

Whether to activate or deactivate the bandwidth profile

Output Parameters

None

Return Values

NSM_VLAN_ERR_IFP_NOT_BOUND when the interface information is not found

NSM_L2_ERR_INVALID_ARG when the port is not configured for switching

NSM_PRO_VLAN_ERR_INVALID_MODE when the port is not configured as a customer edge

NSM_ERR_BW_PROFILE_PER_UNI_CONFIGURED when the bandwidth profile per UNI is configured

NSM_ERR_BW_PROFILE_PER_COS_CONFIGURED when the bandwidth profile per COS is configured

NSM_ERR_BW_PROFILE_NOT_CONFIGURED when the bandwidth profile is not configured

NSM_ERR_BW_PROFILE_CBS_NOT_CONFIGURED when the CBS value is less than MTU

NSM_ERR_BW_PROFILE_EBS_NOT_CONFIGURED when the EBS value is less than MTU

NSM_ERR_BW_PROFILE_EIR_NOT_CONFIGURED when the committed information rate is greater than the excess information rate

NSM_ERR_BW_PROFILE_NOT_ACTIVE when the bandwidth profile is not active

RESULT_OK when the function succeeds

nsm_ovc_bw_profiling

This function configures a bandwidth profile.

Syntax

```
s_int32_t
nsm_ovc_bw_profiling (struct interface *ifp,  struct nsm_ingress_egress_ovc_svid *dual,
                      u_int64_t cir, u_int32_t cbs,  u_int64_t eir, u_int32_t ebs,
u_int8_t coupling_flag, u_int8_t color_mode, u_int8_t bw_profile_type, u_int8_t
bw_profile_parameter)
```

Input Parameters

ifp	Interface pointer
dual	Pointer to nsm_ingress_egress_svid
cir	Committed information rate
cbs	Committed burst rate
eir	Excess information rate
ebs	Excess burst rate
coupling_flag	Whether the coupling flag is enabled or disabled:
PAL_TRUE	Enabled
PAL_FALSE	Disabled
color-mode	Whether color-aware or color-blind

PAL_TRUE Color-aware

PAL_FALSE Color-blind

bw_profile_type

Ingress or egress; one of these constants from nsm/L2/nsm_vlan.h:

NSM_INGRESS_POLICING

NSM_EGRESS_SHAPING

bw_profile_parameter

Bandwidth profile parameter to set; one of these constants from nsm/L2/nsm_vlan.h:

NSM_BW_PARAMETER_CIR

NSM_BW_PARAMETER_CBS

NSM_BW_PARAMETER_EIR

NSM_BW_PARAMETER_EBS

NSM_BW_PARAMETER_COUPLING_FLAG

NSM_BW_PARAMETER_COLOR_MODE

Output Parameters

None

Return Values

NSM_VLAN_ERR_IFP_NOT_BOUND when the interface information is not found

NSM_L2_ERR_INVALID_ARG when the port is not configured for switching

NSM_PRO_VLAN_ERR_INVALID_MODE when the port is not configured as a customer edge

NSM_ERR_BW_PROFILE_PER_UNI_CONFIGURED when the bandwidth profile per UNI is configured

NSM_ERR_BW_PROFILE_PER_OVC_CONFIGURED when the bandwidth profile per OVC is configured

NSM_ERR_COS_ALREADY_CONFIGURED when the COS values are already configured as part of another COS ID

NSM_L2_ERR_MEM NSM_ERR_BW_PROFILE_PER_COS_CONFIGURED when the bandwidth profile configured for the OVC is per COS

NSM_ERR_BW_PROFILE_ACTIVE when the bandwidth profile is active

RESULT_OK when the function succeeds

nsm_delete_ovc_cos_bw_profiling

This function deletes a class of service bandwidth profile.

Syntax

```
s_int32_t
nsm_delete_ovc_cos_bw_profiling (struct interface *ifp,
    struct nsm_ingress_egress_svid *dual,  u_int8_t *cos,
    u_int8_t bw_profile_type, bool_t flag)
```

Input Parameters

ifp Interface pointer

`dual` Pointer to `nsm_ingress_egress_svid`

`cos` Class of service

`bw_profile_type`

Ingress or egress; one of these constants from `nsm/L2/nsm_vlan.h`:

`NSM_INGRESS_POLICING`

`NSM_EGRESS_SHAPING`

`flag` Whether configuring PCPs or DSCPs:

`PAL_TRUE` PCPs

`PAL_FALSE` DSCPs

Output Parameters

None

Return Values

NSM_VLAN_ERR_IFP_NOT_BOUND when the interface information is not found

NSM_L2_ERR_INVALID_ARG when the bandwidth profile is not found

NSM_PRO_VLAN_ERR_INVALID_MODE when the port is not configured as a customer edge

NSM_ERR_BW_PROFILE_PER_UNI_CONFIGURED when the bandwidth profile per UNI is configured

NSM_ERR_BW_PROFILE_PER_OVC_CONFIGURED when the bandwidth profile per OVC is configured

NSM_ERR_COS_ALREADY_CONFIGURED when the COS values are already configured as part of another COS ID

NSM_ERR_BW_PROFILE_NOT_CONFIGURED when the bandwidth profile is not configured

NSM_ERR_BW_PROFILE_ACTIVE when the bandwidth profile is active

RESULT_OK when the function succeeds

nsm_set_ovc_cos_bw_profiling

This function activates or deactivates a class of service (CoS) bandwidth profile.

Syntax

```
s_int32_t
nsm_set_ovc_cos_bw_profiling (struct interface *ifp,
    struct nsm_ingress_egress_svid *dual, u_int8_t *cos,
    u_int8_t bw_profile_type, u_int8_t bw_profile_status, bool_t flag)
```

Input Parameters

`ifp` Interface pointer

`dual` Pointer to `nsm_ingress_egress_svid`

`cos` Class of service values

`bw_profile_type`

Ingress or egress; one of these constants from `nsm/L2/nsm_vlan.h`:

`NSM_INGRESS_POLICING`

`NSM_EGRESS_SHAPING`

`bw_profile_status`
: Whether to activate or deactivate the bandwidth profile

`flag`
: Whether to activate PCPs or DSCPs:

`PAL_TRUE`
: PCPs

`PAL_FALSE`
: DSCPs

Output Parameters

None

Return Values

NSM_VLAN_ERR_IFP_NOT_BOUND when the interface information is not found

NSM_L2_ERR_INVALID_ARG when:

- `dual`, `cos`, or `bw_profile_type` is NULL
- A VLAN is not associated to the OVC
- The port is not configured for switching

NSM_PRO_VLAN_ERR_INVALID_MODE when the port is not configured as a customer edge

NSM_ERR_BW_PROFILE_PER_UNI_CONFIGURED when the bandwidth profile per UNI is configured

NSM_ERR_BW_PROFILE_PER_OVC_CONFIGURED when the bandwidth profile per OVC is configured

NSM_COS_ID_DSCP_ALREADY_CONFIGURED when the CoS is configured for DSCP, but `flag` specifies PCP

NSM_COS_ID_ALREADY_CONFIGURED_ERR when PCP is already active

NSM_INVALID_COS_ID_ERR when `cos` is not valid

NSM_COS_ID_PCP_ALREADY_CONFIGURED when the CoS is configured for PCP, but `flag` specifies DSCP

NSM_ERR_COS_ALREADY_CONFIGURED when the COS values are already configured as part of another CIS ID

NSM_ERR_BW_PROFILE_PER_OVC_CONFIGURED when CoS values are already configured as part of another CoS identifier

NSM_ERR_BW_PROFILE_NOT_CONFIGURED when the bandwidth profile is not configured

NSM_ERR_BW_PROFILE_CBS_NOT_CONFIGURED when the CBS value is less than the MTU

NSM_ERR_BW_PROFILE_EBS_NOT_CONFIGURED when the EBS value is less than the MTU

NSM_ERR_BW_PROFILE_EIR_NOT_CONFIGURED when the committed information rate is greater than the excess information rate

NSM_ERR_BW_PROFILE_NOT_ACTIVE when the bandwidth profile is not active

RESULT_OK when the function succeeds

nsm_ovc_cos_bw_profiling

This function configures a class of service bandwidth profile.

Syntax

```
s_int32_t
nsm_ovc_cos_bw_profiling (struct interface *ifp,
    struct nsm_ingress_egress_svid *dual, u_int8_t *cos, u_int64_t cir,
```



```
u_int32_t cbs, u_int64_t eir, u_int32_t ebs, u_int8_t coupling_flag,
u_int8_t color_mode, u_int8_t bw_profile_type,
u_int8_t bw_profile_parameter, bool_t flag
```

Input Parameters

ifp	Interface pointer
dual	Pointer to nsm_ingress_egress_svid
cos	Class of service
cir	Committed information rate
cbs	Committed burst rate
eir	Excess information rate
ebs	Excess burst rate
coupling_flag	Whether the coupling flag is enabled or disabled:
PAL_TRUE	Enabled
PAL_FALSE	Disabled
color-mode	Whether color-aware or color-blind:
PAL_TRUE	Color-aware
PAL_FALSE	Color-blind
bw_profile_type	Ingress or egress; one of these constants from nsm/L2/nsm_vlan.h:
NSM_INGRESS_POLICING	
NSM_EGRESS_SHAPING	
bw_profile_parameter	Bandwidth profile parameter to set; one of these constants from nsm/L2/nsm_vlan.h:
NSM_BW_PARAMETER_CIR	
NSM_BW_PARAMETER_CBS	
NSM_BW_PARAMETER_EIR	
NSM_BW_PARAMETER_EBS	
NSM_BW_PARAMETER_COUPLING_FLAG	
NSM_BW_PARAMETER_COLOR_MODE	
flag	Whether configuring PCPs or DSCPs:
PAL_TRUE	PCPs
PAL_FALSE	DSCPs

Output Parameters

None

Return Values

NSM_VLAN_ERR_IFP_NOT_BOUND when the interface information is not found

NSM_L2_ERR_INVALID_ARG when:

- `dual`, `cos`, or `bw_profile_type` is NULL
- A VLAN is not associated to the OVC
- The port is not configured for switching

NSM_PRO_VLAN_ERR_INVALID_MODE when the port is not configured as a customer edge

NSM_COS_ID_DSCP_ALREADY_CONFIGURED when the CoS is configured for DSCP, but `flag` specifies PCP

NSM_COS_ID_ALREADY_CONFIGURED_ERR when PCP is already active

NSM_INVALID_COS_ID_ERR when `cos` is not valid

NSM_COS_ID_PCP_ALREADY_CONFIGURED when the CoS is configured for PCP, but `flag` specifies DSCP

NSM_ERR_COS_ALREADY_CONFIGURED when the CoS values are already configured as part of another CIS ID

NSM_ERR_BW_PROFILE_PER_OVC_CONFIGURED when CoS values are already configured as part of another CoS identifier

NSM_ERR_CIR_BW_PROFILE when the CIR for all classes of service exceeds the interface bandwidth

NSM_ERR_EIR_BW_PROFILE when the EIR for all classes of service exceeds the interface bandwidth

NSM_ERR_BW_PROFILE_NOT_CONFIGURED when the bandwidth profile is not configured

NSM_ERR_BW_PROFILE_PER_UNI_CONFIGURED when the bandwidth profile per UNI is configured

NSM_ERR_BW_PROFILE_PER_OVC_CONFIGURED when the bandwidth profile per OVC is configured

NSM_ERR_COS_ALREADY_CONFIGURED when the CoS values are already configured as part of another CoS ID

NSM_L2_ERR_MEM when memory allocation for CoS bandwidth profile fails

NSM_ERR_BW_PROFILE_ACTIVE when the bandwidth profile is active

RESULT_OK when the function succeeds

Appendix A G.8032 (ERPS) Version 1

This appendix contains the Ethernet Ring Protection Switching (ERPS) version 1 application programming interface. These functions are obsolete and are included here for reference.

ZebOS-XP supports multiple Ethernet Ring instances, in accordance with the ITU-T G.8032 (2008) Ethernet Ring Protection OAM and APS Model recommendation.

Overview

Ethernet rings provide wide-area multipoint connectivity more economically due to the reduced number of links required. A single port on a ZebOS-XP bridge instance can support both ring and non-ring port operation on a per-VLAN basis. The definition of an Ethernet ring is applicable to both physical ring topologies and logical ring topologies. The mechanisms and protocol defined in the ITU-T G.8032 recommendation achieve highly reliable and stable protection, and never form loops that would fatally affect network operation and service availability. Protection switching (PS) is agnostic to the monitoring method used, as long as it can receive OK (success) or SF (switch failure) information for the transport entity of each ring link.

The fundamentals of a ring protection switching architecture are the principle of loop avoidance and the utilization of learning, forwarding, and address table mechanisms defined in the Ethernet flow forwarding function (ETH_FF). Loop avoidance in a ring is achieved by guaranteeing that, at any time, traffic may flow on all but one of the ring links. This link is called the Ring Protection Link (RPL), and under normal conditions it is blocked (not used for traffic). One designated node, the RPL Owner, is responsible for blocking traffic on the RPL. In the event of a ring failure, the RPL Owner is responsible for unblocking the RPL and allowing it to be used for traffic. Ethernet ring networks may support only one active running loop avoidance algorithm or protocol, for example STP/RSTP/MSTP, or Ethernet Ring Protection (ERP) at a time.

Features

- Ethernet Ring Protection prevents the creation of loops in a ring topology under all circumstances, including network start up, failure detection, or switchover.
- Ethernet layer connectivity of ring links is periodically monitored to inform the ERP mechanism of Switch Fail (SF) or Switch Down (SD) conditions, link bandwidth degradation or excessive errors. Server Layer SF and SD conditions are also reported to ERP.
- ERP does not contend with protection mechanisms of the service layer, and is independent of the capability of the server layer. The total communication bandwidth consumed by the protection mechanism is a very small fraction of the total available bandwidth, and is independent of the total traffic supported by the network.
- A ring can successfully recover multipoint connectivity in the event of a single ring link failure, and multipoint connectivity in the event of a single node failure, except for the traffic at that node. In the event of more than a single failure, ring segmentation with full connectivity within each segment and ERP operates under all network load conditions.
- ERP supports protection switching over multi-ring or ladder networks. The protection mechanism enables interconnection of rings using a single node or dual nodes (a shared link), and protects services that are traversing interconnected rings. In the case of interconnected rings using dual nodes, ERP ensures that a super loop is not formed in the event that a shared link fails.
- Protection mechanism does not impose any limitation or requirements on the Ethernet relay and filtering function, nor does it impose any limitation on the number of nodes that may form an Ethernet ring. From an operational perspective, the maximum number of nodes supported ranges from 16 to 255.

- In the event of a single ring node or link failure, an administratively-triggered switchover, or reversion, ERP supports a protection switching time (transfer time) of no more than 50ms, when the ring size is less than or equal 16 nodes, and there is no traffic on the ring other than Ring Automated Protection Switching (R-APS) messages.
- ERP has configurable hold-off times before triggering a protection operation and configurable wait-to-restore times.
- Ethernet ring nodes support MAC services and QoS according to the IEEE 802.1Q specification.
- Ethernet ring topologies support unicast, multicast, and broadcast communication.

Architecture

These ZebOS-XP modules support ERPS:

- NSM
- ONMD/CFM
- MSTPD

The NSM module maintains centralized bridge, VLAN, and interface configuration information that is distributed to the ONMD module. The ONMD module supports Connectivity Check Messaging (CCM) and Automated Protection Switching (APS) messaging, plus the state machines that are driven by SF and other events to sustain protection behaviors.

Terms

Table 1: Terms and Concepts

Term	Description
Ring Protection Link (RPL)	Link designated by ERPS that is blocked during the Idle state to prevent loops on a Bridged ring.
RPL Owner	Node connected to RPL that blocks traffic on the RPL while it is in the Idle state and unblocks traffic while it is in the Protected state.
Link Monitoring.	Link monitoring is managed using standard Ethernet Connectivity Check (CC) OAM messages.
Signal Fail (SF)	Signal Fail is declared when an Ethernet trail signal fail condition is detected.
No Request (NR)	No Request is declared when there are no outstanding conditions, for example, SF, on the node.
Ring APS (R-APS) Messages	Protocol messages defined in the ITU-T Y.1731 and G.8032 recommendations.
Automatic Protection Switching (APS) Channel	Ring-wide VLAN used exclusively for transmission of OAM messages including R-APS messages

Timers

Table 2: Timer

Timer	Description
TU-T G.8032	specifies the use of two timers to avoid race conditions and unnecessary switching operations.

Table 2: Timer (Continued)

Timer	Description
WTR (Wait to Restore) Timer	Used by the RPL Owner to verify that the ring has stabilized before blocking the RPL after SF Recovery
Hold-off Timer	Used by underlying Ethernet layer to filter out intermittent link faults.Faults will only be reported to the ring protection mechanism if this timer expires.

Note: In ZebOS-XP, the on and off functions for the timers are controlled by the l2_start_timer() and l2_stop_timer() functions.

Principles of Operation

The protection algorithm is based on the transmission of local switch requests and status information to all ring nodes using R-APS messages.

Controlling the Protection Mechanism

Protection switching is triggered by the detection or clearing of Signal Failure via Ethernet CCM, remote requests over the R-APS channel, or expiration of the timers. R-APS requests control the communication and states of a ring node. Two basic R-APS messages are specified: R-APS(SF) and R-APS(NR). The RPL owner may modify the R-APS(NR) to indicate that the RPL is blocked, for example, R-APS(NR, RB).

Normal Operation

Ring nodes in a normal operational state have the characteristics illustrated below.

- No link or node faults have been detected in the ring.
- The Physical topology has all nodes connected in a ring.
- ERP guarantees no loops by blocking the RPL.
- The Logical topology has all nodes connected without a loop.
- Each link is monitored by its two adjacent nodes using Ethernet Connectivity Check messages.
- Signal failure, loss of continuity, or server layer failure triggers ring protection.

Protecting Operation

Ring nodes can be in protection mode. This means protection switching is in effect after a signal fault was identified.

Interconnected Rings

Characteristics of interconnected rings include the following:

- Ring interconnection should not require changes to the single ring protection mechanism.
- Nodes that are not at the ends of shared links do not need special provisioning to support shared links in the ring.
- When a shared link fails, it is necessary to prevent the formation of a super loop, which may happen if both rings protect at the same time.
- A shared link may act as the RPL for only one of the rings that share the link.
- A signal failure on a non-shared link (when the ring is in idle state) should only trigger protection switching within the ring where the link failed; other interconnected rings should be agnostic to this event.
- Interconnected rings may share more than one shared link. A shared link may be shared by more than two rings.

- A node that is common to different rings may be connected by more than one shared link.
- Two rings may not be connected by two shared nodes if the link between these nodes is not shared, for example, when two links exist between the two shared nodes.
- Two rings may not be connected by any two shared links if the links are not connected to the same shared node.
- When a shared link fails, only RPL with highest priority protects ring. This prevents the formation of super loop.

Signaling Channel. ERP uses R-APS messages to manage and coordinate the protection switching. Each ring is addressed separately in R-APS messages using information defined in ITU-T Y.1731. OAM common fields are also defined in ITU-T Y.1731.

- **Version** 00000 for this version of Recommendation
- **OpCode** Defined as 40 for R-APS in Y.1731
  - **Flags** 00000000; are ignored by ERP

R-APS Specific Information

The following is a breakdown of the R-APS specific portion of the ETH-OAM message.

- **Request/Status** (4bits) 1011 = SF | 0000 = NR | Other = Future
- **Status RB** (1bit) Set when RPL is blocked (used by RPL Owner in NR).
- **Status DNF** (1bit) Set when FDB Flush is not necessary (future development).
- **Node ID** (6octets) MAC address of message source node (informational).
- **Reserved1** (4bits), Status Reserved (6 bits), Reserved2 (24octets) (for future development).

Command API

The functions in this section are called by the commands in the *Carrier Ethernet Command Reference*.

Function	Description
g8032_api_update_rpl	Updates the ring protection link owner
g8032_api_update_shared_link	Updates shared link information for protection rings
g8032_api_update_timers	Updates the timers for a protection ring
g8032_cfm_association	Adds a CFM association and configures an MD name, an MA name, and a bridge name
g8032_cfm_de_association	Deletes a CFM association from an MD, an MA, and a bridge
g8032_find_ring_by_id	Gets a ring node

Include Files

To use the functions in this chapter, you need to include `onmd/cfm/g8032/g8032_api.h`.

g8032_api_update_rpl

This function updates the ring protection link owner.

Syntax

```
s_int32_t
g8032_api_update_rpl (u_int8_t *br_name,
                      u_int32_t ring_id,
                      bool_t is_east,
                      u_int8_t non_owner_set_val,
                      u_int32_t node_type)
```

Input Parameters

`br_name`	Bridge name on which the protection ring exists
`ring_id`	Protection ring ID
`is_east`	Whether the node is east.
`non_owner_set_val`	Value of non-owner status
`node_type`	Type of node affected

Output Parameters

None

Return Values

G8032_RING_NULL when the ring instance is not found

G8032_INTERFACE_MISMATCH when there is a mismatch of the interface configured with the ring

G8032_CFM_MEP_NOTFOUND when the MEP is not found

G8032_FAILURE when the function fails

G8032_API_ERR_NONE when the function succeeds

g8032_api_update_shared_link

This function updates shared link information for protection rings.

Syntax

```
s_int32_t
g8032_api_update_shared_link (struct g8032_ring_node *ring,
                              u_int32_t ifindex,
                              u_int32_t peer_ring_id)
```

Input Parameters

`ring`	Pointer to the ring node
`ifindex`	Index value of the interface
`peer_ring_id`	Peer protection ring ID <1-65535>

Output Parameters

None

Return Values

G8032_RING_NULL when the ring instance is not found

G8032_SHARED_LINK_ON_RPL when the RPL port cannot be configured as a shared link

G8032_SHARED_LINK_CNTRL_VLAN_CONFLICT when the primary VLAN for the ring and peer ring do not match

G032_CONFIG_COMPLETE when configuration is not complete

G8032_SHARED_LINK_NOT_CONFG when the shared link is not configured

G8032_API_ERR_NONE when the function succeeds

g8032_api_update_timers

This function updates the timers for a protection ring.

Syntax

```
s_int32_t
g8032_api_update_timers (u_int8_t *br_name,
                         u_int32_t ring_id,
                         u_int32_t type,
                         u_int32_t timeout)
```

Input Parameters

`br_name`	Bridge name on which the protection ring exists
`ring_id`	Protection ring ID
`type`	Type of timer to reset
`timeout`	Value for timer

Output Parameters

None

Return Values

G8032_API_ERR_RING_NOT_FOUND when the ring is not found

G8032_INVALID_GUARD_TIMEOUT when the guard timeout is not in multiples of 10 milliseconds

G8032_INVALID_HOLD_OFF_TIMEOUT when the hold off timeout is not in multiples of 10 seconds

G8032_INVALID_WTR_TIMEOUT when the wait-to-restore timeout is not in a multiple of 1 minute

G8032_WTR_NOT_ALLOWED_FOR_NON_OWNER when the wait-to-restore can only be configured on an RPL owner node

G8032_FAILURE when the function fails

G8032_API_ERR_NONE when the function succeeds

g8032_cfm_association

This function adds a CFM association and configures an MD name, an MA name, and a bridge name.

Syntax

```
s_int32_t
g8032_cfm_association (u_int8_t *md_name,
                       u_int8_t *ma_name,
                       u_int8_t *br_name,
                       u_int16_t ring_id)
```

Input Parameters

md_name	Maintenance domain name
ma_name	Maintenance association name
br_name	Bridge name on which the protection ring exists
ring_id	Protection ring ID

Output Parameters

None

Return Values

G8032_BRIDGE_NOT_FOUND when the bridge is not found

G8032_RING_NOT_FOUND when the ring is not found

G8032_CFM_MD_NOTFOUND when the MD is not found

G8032_CFM_MA_NOTFOUND when the MA is not found

G8032_PVID_OF_MA_PG_NOT_MATCHING then the primary VID of the ring and the MA did not match

G8032_ONMD_BR_PORT_NOT_FOUND When the bridge port is not found

CFM_API_ERR_IF_NOT_FOUND if the interface is not found

G8032_CFM_MEP_NOTFOUND when the MEP is not found

G8032_MEP_MISTMATCH when there is a mismatch in MEPs

G8032_API_ERR_NONE when the function succeeds

g8032_cfm_de_association

This function deletes a CFM association from an MD, an MA, and a bridge.

Syntax

```
s_int32_t
g8032_cfm_de_association (u_int8_t *md_name,
                          u_int8_t *ma_name,
                          u_int8_t *br_name,
                          u_int16_t ring_id)
```

Input Parameters

md_name	Maintenance domain name
ma_name	Maintenance association name
br_name	Bridge name on which the protection ring exists
ring_id	Protection ring ID

Output Parameters

None

Return Values

G8032_BRIDGE_NOT_FOUND when the bridge is not found

G8032_RING_NOT_FOUND when the ring is not found

G8032_CFM_MEP_NOTFOUND when the MEP is not found

G8032_API_ERR_NONE when the function succeeds

g8032_find_ring_by_id

This function gets a ring node.

Syntax

```
struct g8032_ring_node *
g8032_find_ring_by_id (u_int8_t * br_name, s_int32_t ring_id)
```

Input Parameters

`br_name`	Bridge name on which the protection ring exists
`ring_id`	Protection ring ID

Output Parameters

None

Return Values

Pointer to the `g8032_ring_node` structure when the function succeeds

NULL when the bridge name is not valid

Index

Numerics

A

B

C

D

E

F

I

L

M

N

O

P

R

S

T

U

V

W