ZebOS-XP®
Network Platform

Version 1.4
Extended Performance

Multi-Protocol Label Switching
Developer Guide

December 2015

IP Infusion Inc. Proprietary

© 2015 IP Infusion Inc. All Rights Reserved.

This documentation is subject to change without notice. The software described in this document and this documentation are furnished under a license agreement or nondisclosure agreement. The software and documentation may be used or copied only in accordance with the terms of the applicable agreement. No part of this publication may be reproduced, stored in a retrieval system, or transmitted in any form or any means electronic or mechanical, including photocopying and recording for any purpose other than the purchaser's internal use without the written permission of IP Infusion Inc.

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

Trademarks:
IP Infusion, OcNOS, VirNOS, ZebM, ZebOS, and ZebOS-XP are trademarks or registered trademarks of IP Infusion. All other trademarks, service marks, registered trademarks, or registered service marks are the property of their respective owners.

Contents

© 2015 IP Infusion Inc. Proprietary

© 2015 IP Infusion Inc. Proprietary

© 2015 IP Infusion Inc. Proprietary

© 2015 IP Infusion Inc. Proprietary

© 2015 IP Infusion Inc. Proprietary

© 2015 IP Infusion Inc. Proprietary

© 2015 IP Infusion Inc. Proprietary

© 2015 IP Infusion Inc. Proprietary

© 2015 IP Infusion Inc. Proprietary

© 2015 IP Infusion Inc. Proprietary

Preface

This guide describes the ZebOS-XP application programming interface (API) for Multi-Protocol Label Switching (MPLS).

Audience

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

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 guide contains these chapters and appendices:

- Chapter 1, *Multi-Protocol Label Switching Overview*
- Chapter 2, *Virtual Private LAN Service*
- Chapter 3, *BGP Virtual Private LAN Services*
- Chapter 4, *MPLS Transport Profile*
- Chapter 5, *Layer 2 Virtual Private Network*
- Chapter 6, *MPLS-TP OAM Modules*
- Chapter 7, *MPLS-TP OAM Y-1731 Support*
- Chapter 8, *MPLS-TP Linear Protection Switching*
- Chapter 9, *MPLS-TP Ring Protection Switching*
- Chapter 10, *MPLS Non-Stop Forwarding*
- Chapter 11, *Layer 2 Virtual Circuit*
- Chapter 12, *DiffServ-TE*
- Chapter 13, *Structure Agnostic TDM over Packet*

Related Documents

The following guides are related to this document:

- *Architecture Guide*
- *Multi-Protocol Label Switching Command Reference*
- *Multi-Protocol Label Switching Configuration Guide*
- *Multi-Protocol Label Switching Software Forwarder Developer Guide*
- *Label Distribution Protocol Command Reference*
- *Label Distribution Protocol Developer Guide*
- *RSVP-TE Command Reference*
- *RSVP-TE Developer 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.

© 2015 IP Infusion Inc. Proprietary

CHAPTER 1 Multi-Protocol Label Switching Overview

The ZebOS-XP Multi-Protocol Label Switching (MPLS) modules provide a complete solution for the rapid integration of MPLS functionality into enterprise, edge, and core communications equipment. MPLS operates between the Data Link layer (Layer 2) and the Network layer (Layer 3) in the standard OSI model. MPLS provides a unified data-carrying mechanism for both circuit-based and packet-switching clients.

Traffic Engineering (TE) extensions use the Resource ReServation Protocol-Traffic Engineering (RSVP-TE) dynamic signaling protocol to communicate to the ZebOS-XP MPLS Forwarder or to a third-party MPLS forwarder. The RSVP-TE extension enables MPLS to scale into large IP-based communications equipment.

Features

- Label Switched Path (LSP) setup using Label Distribution Protocol LDP
- Traffic-Engineered LSP setup using RSVP-TE
- Layer 3 Virtual Private Networks (VPN) over MPLS core
- Virtual Circuits over MPLS core
- Virtual Private LAN Services (VPLS) using LDP signaling
- BGP-VPLS Signaling
- MPSL Differentiated Services (DiffServ)
- MPLS Operations, Administration and Management (OAM), including
  - Detecting MPLS Data Plane Failures
  - Pseudowire Virtual Circuit Connectivity Verification (VCCV)
  - Bidirectional Forwarding Detection (BFD) for VCCV
  - BFD for MPLS LSPs
- A Linux-based MPLS Forwarder
- Constrained Shortest Path First (CSPF) computation
- Graceful Restart for LDP and RSVP-TE
- Fast Reroute, with One-to-One and Facility protections
- RSVP Hello support
- RSVP-TE Refresh Reduction
- Support for full mesh VPLS networks
- Virtual Routing (VR) and Virtual Route Forwarding (VRF)
- MPLS MIB
- MPLS Traffic Engineering MIB
- MPLS Label Switch Router (LSR) MIB
- Pseudowire (PW) Setup using LDP
- PW Emulation Edge-to-Edge (PWE3)
- PW MIB

- PW Ethernet MIB
- MPLS-Transport Profile (MPLS-TP)
- MPLS-TP Linear Protection Switching
- MPLS-TP OAM, including
  - Fault Measurement
  - Loss Delay Measurement
  - Lock Instruct and Loopback

© 2015 IP Infusion Inc. Proprietary

CHAPTER 2 Virtual Private LAN Service

Virtual Private LAN Service (VPLS) is a way to provide Ethernet-based multipoint-to-multipoint communication over IP/MPLS networks. VPLS is Virtual Private Network (VPN) technology that supports any-to-any connectivity, in contrast to L2TPv3, which only supports point-to-point Layer 2 tunnels.

VPLS Overview

Virtual private LAN service provides a way to enable transparent Layer 2 Ethernet LAN services to geographically dispersed customer sites connected by a Wide Area Network (WAN) by providing support for traditional Layer 2 broadcast and multicast services.

Customer sites are connected through pseudowires (PW) to share an Ethernet broadcast domain that provides Ethernet based multipoint-to-multipoint communication over IP or MPLS networks. VPLS uses a set of Martini circuits, grouped by a common VPLS identifier, to achieve this service objective. A PW consists of a pair of point-to-point single-hop unidirectional LSPs configured in opposite directions, each identified by a PW label, also called a Virtual Connection (VC) label.

The Label Discovery Protocol (LDP) is used to signal constituent VCs and the service provider may use LDP, RSVP-TE, or add static provisioning to set up LSP tunnels to transport data through virtual circuits (VC). The VPLS identifier is exchanged with labels, so that both PWs can be linked and associated with a particular VPLS instance. The ZebOS-XP VPLS implementation supports both full mesh and hub-and-spoke topologies for VPLS networks.

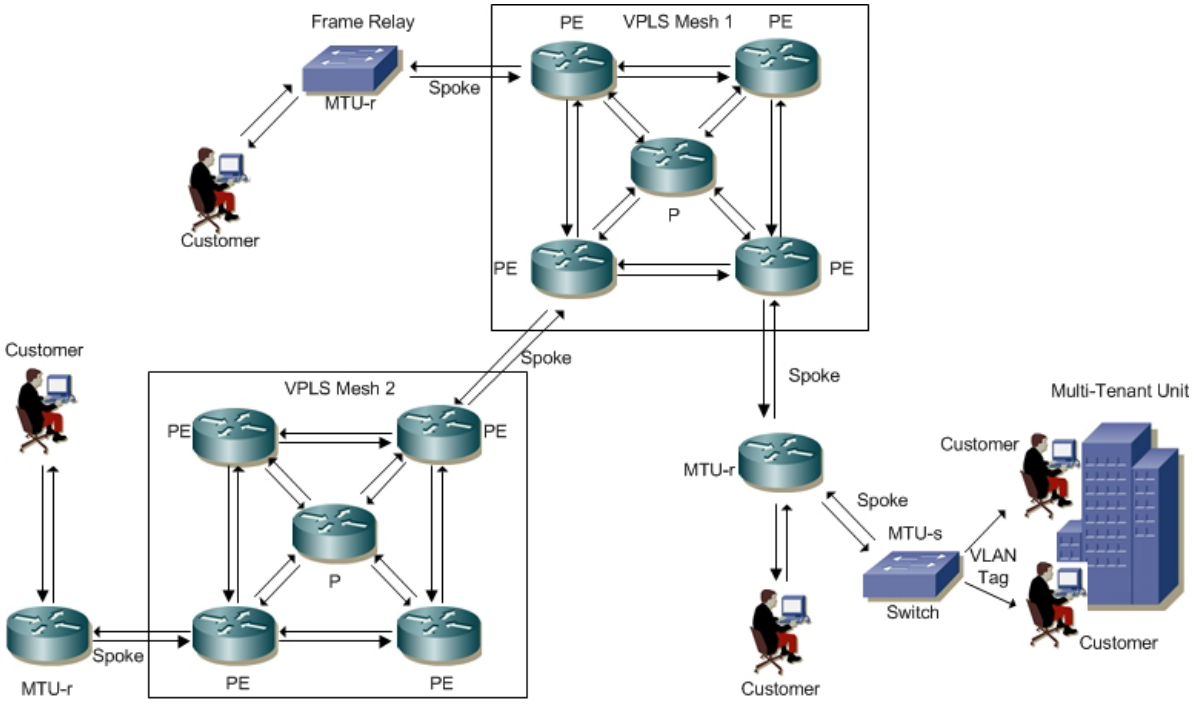

Figure 2-1: Illustration of Network with VPLS

Features

The ZebOS-XP VPLS features are described below.

Full Mesh VPLS Topology

In a Full Mesh configuration, a virtual circuit is created between all the PE (Provider Edge) routers participating in a VPLS instance. A Mesh VC refers to a VC created between two PE routers participating in a Full Mesh configuration. Each PE router can support multiple simultaneous VPLS instances, which means that each PE router can participate in multiple Layer-2 VPNs at the same time. One caveat for Full-Mesh topologies is that data received from a PE router must not be forwarded to another PE router to avoid loops (split horizon).

Hub-and-Spoke Topology

In a Hub-and-Spoke topology, some routers are designated as hub/core PE routers and other routers are designated as spokes. The hub nodes form a full VC Mesh similar to Full Mesh topology. The Spoke routers need to have connectivity to only one hub router and need not participate in full mesh signaling for the VPLS instance that they are participating in. A Spoke VC may connect to more than one core PE router but only one spoke connection is designated as a Primary Spoke and is used for data forwarding. The Secondary Spoke connection is activated on failure of Primary Spoke connection.

NSM Extensions

The ZebOS-XP Network Services Module (NSM) provides commands you use to configure VPLS instances. Messages are exchanged between LDP and NSM that convey VPLS information to the LDP module.

The VPLS add message is sent when either a new VPLS instance is created or an existing instance is modified. The VPLS add message is sent only if at least one mesh peer is configured for that instance. This message is also generated when a mesh peer is either added or deleted.

The VPLS delete message is sent to LDP when either a configured VPLS instance is deleted or all VPLS mesh peers are deleted from the VPLS configuration.

An existing MPLS Layer 2 Circuit may be added to a VPLS instance to act as a spoke VC. To act as a spoke VC, this MPLS L2 VC should not be already bound to an interface, because an interface cannot be bound to VRF, VC and VPLS at the same time

LDP Extensions

A targeted peer must be configured for each Hub or Spoke peer configured in NSM to create VPLS mesh or Spoke Virtual Circuits. The ZebOS-XP LDP module supports VPLS configuration by processing the VPLS add or delete messages received from NSM.

A VPLS table is located in LDP to store all VPLS instances, along with mesh virtual circuits. All mesh VCs belonging to a VPLS must have the same VC ID.

A Targeted LDP session is created with all configured spoke and mesh peers in DU mode with liberal retention option. This session is used for VPLS signaling. Same session can be used to signal multiple Mesh/Spoke VPLS Virtual Circuits.

Note: You need to explicitly configure targeted peers in the Router mode running in the LDP daemon.

 © 2015 IP Infusion Inc. Proprietary

Sample VPLS Network

A sample network is used to explain VPLS implementation. In this example, a company, Company One has several sites. All sites of this company are connected together using one VPLS configured over several types of devices. The PE devices form the VPLS mesh and are VPLS mesh peers of each other. Customer sites can connect directly to PE devices that make up the VPLS mesh. To avoid full mesh connectivity, some PE devices connect using a hierarchical model to the spoke devices MTU-r and MTU-s. Spoke connectivity eliminates the need for a full mesh of tunnels and full mesh of connections per service between all devices participating in the VPLS.

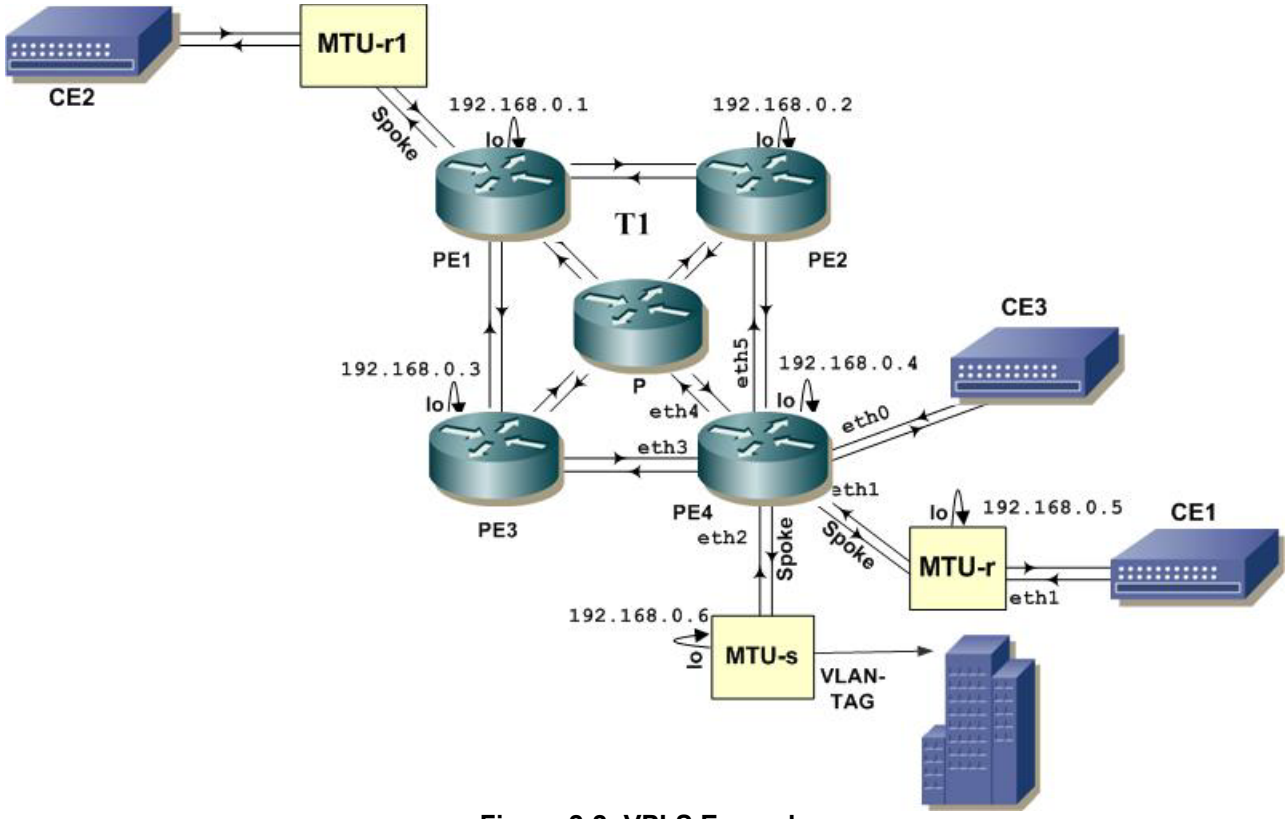

Figure 2-2: VPLS Example

In this illustration, various departments for the Customer (represented by the building) connect to the VPLS through the bridging-capable MTU-s device. The MTU-s device assigns VLAN tags to the various sites connecting to itself. The MTU-s device connects to PE4 (a PE device) using a single point-to-point VC. Customer sites CE1 and CE2 are connected to the VPLS mesh through the MTU-r devices, which are not bridging capable. The MTU-r device has one VC for every site that must be joined to the VPLS.

Consider a frame that needs to be sent from a host located in the Company site to a host at the CE2 site. The frame is first transported over the spoke VC to the PE4 device where it is labeled for transport over the Provider MPLS cloud (P). Transport LSPs are preconfigured in the cloud using RSVP-TE or LDP. The labelled packet is transported to PE1, and from PE1 to the MTU-r device. At the MTU-r device the transport frame is delivered to the appropriate interface based on the VC label in the frame.

© 2015 IP Infusion Inc. Proprietary

CHAPTER 3 BGP Virtual Private LAN Services

The Virtual Private LAN Services (VPLS) architecture defines a mechanism to offer multipoint Ethernet services over a shared MPLS infrastructure. To achieve this, every VPLS instance in the network simulates the behavior of an IEEE 802.1D bridge. This is done by setting up point-to-point pseudowires (PW) between a node and every other node in the VPLS instance, thereby creating a full mesh of PWs between all nodes in the VPLS instance. The full mesh of PWs that is created ensures that any node can reach any other node in the VPLS using a single PW hop. BGP VPLS as specified in RFC 4761 is used to auto-discover the peer nodes and signal labels between PW end points.

Overview

To configure and manage VPLS with BGP auto-discovery plus signaling, or LDP-based signaling with BGP auto-discover, a command sub-mode, called signaling, is implemented in MPLS VPLS. The signaling sub-mode of MPLS VPLS handles both BGP-VPLS signaling and LDP-VPLS signaling.

Once a VPLS is configured, NSM is informed and forwards a message to either BGP or LDP. BGP handles the creation of new VPLs instances, peer discovery for the L2VPN address family, and signaling using NLRI messages. BGP also interacts with NSM to get label block information for VPLS signaling. When PW signaling is complete, BGP gives a command to NSM to add the VC FIB entries for each PW. VPLS auto-discovery enables VPLS PE routers to discover other PE routers that are part of the same VPLS domain. Signaling also sets up or tears down pseudowires with other PE routers.

Auto-Discovery

Discovery refers to the process of finding all PEs that participate in a VPLS instance. This allows each PE configuration to consist of only the identity of the VPLS instance established on the PE, not the identity of every other PE in the VPLS instance. When the topology of a VPLS changes, only the affected PE configuration changes and all other PEs are automatically notified of the change and adapt.

BGP uses the L2VPN Routing Information Base (RIB) to store endpoint provisioning information, which is updated each time any Layer 2 VPLS instance is configured. Prefix and path information is stored in the L2VPN database, allowing BGP to determine the best path. When BGP distributes endpoint provisioning information in an update message to all of its neighbors, the endpoint information is used to configure a pseudowire mesh to support L2VPN-based services.

Signaling

Signaling is, at its most basic, the set up and tear down of pseudowires. Once discovery is accomplished, each pair of PEs in a VPLS must be able to establish and tear down pseudowires with each other. Signaling is also used to transmit characteristics of the pseudowires that a PE sets up for a VPLS.

A demultiplexer is used to distinguish among different streams of traffic carried over a tunnel, each stream possibly representing a different service. In the case of VPLS, the demultiplexer not only determines to which specific VPLS a packet belongs, but also identifies the ingress PE. The first part of this information is used for forwarding the packet; the second part is used for learning MAC addresses. The demultiplexer used here is an MPLS label.

Using a distinct BGP update message to send a demultiplexer to each remote PE would require the originating PE to send X such messages to Y remote PEs. The solution described in RFC 4761 allows a PE to send a single (common) update message that contains demultiplexers for all remote PEs, instead of X individual messages.

The following figure illustrates the process of VPLS configuration with BGP auto-discovery and signaling.

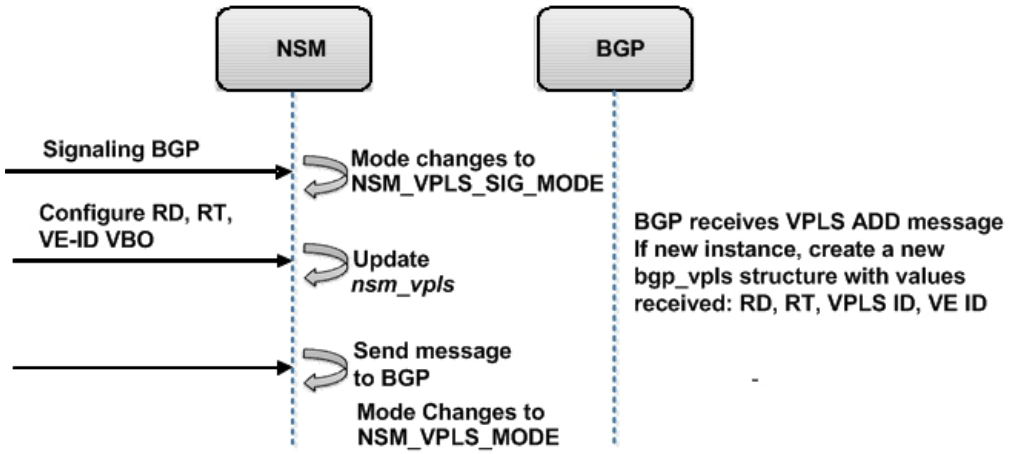

Figure 3-1: A PE configured for auto-discovery and VPLS

When you enter the command `signaling bgp`, the mode is changed from `NSM_VPLS_MODE` to `NSM_VPLS_SIG_MODE`. In the signaling mode, you configure Route Distinguisher (RD), Route target (RT), VE ID (VPLS edge device) and VPLS range for a VPLS instance `V`. When configuration is complete, the `nsm_vpls` structure is updated with RD, RT, VE ID, and VPLS range values in NSM.

When you use the `signaling bgp` command, the auto-discovery mode is enabled by default. In this manner BGP is used for both auto-discovery and signaling of VPLS. When signaling is effected with LDP, you need to explicitly enable the BGP auto-discovery (`bgp-ad`) parameter in the signaling command.

BGP Address Family

When you issue the command `address-family l2vpn vpls`, the mode is changed from the BGP router mode to BGP VPN mode. In this mode, you configure the neighbor and the network for the VPLS instance. Once configuration is done, BGP updates its structure with the address family information.

Note: See the *Border Gateway Protocol Command Reference* for details about the `address-family` command.

© 2015 IP Infusion Inc. Proprietary

CHAPTER 4 MPLS Transport Profile

Multi-Protocol Label Switching - Transport Profile (MPLS-TP) is a simplified subset of MPLS that supports the following transport-type functions:

- Point-to-point unidirectional tunnels
- Point-to-point associated tunnels
- Point-to-point co-routed tunnels
- IP-based identifiers for MPLS TP identifiers
- LSP static provisioning of transport paths via the management plane
- BFD-MPLS or data link OAM static provisioning of transport paths via the management plane
- Supported transport services: IPv4 traffic and pseudowires
- LSPs are manually set up
- Protection switching is driven by OAM monitors

MPLS-TP excludes the following functions of MPLS:

- Penultimate Hop Popping (PHP)
- Label-Switched Paths (LSPs) merge
- Equal Cost Multi Path (ECMP)
- MPLS-TP does not require MPLS control plane capabilities.

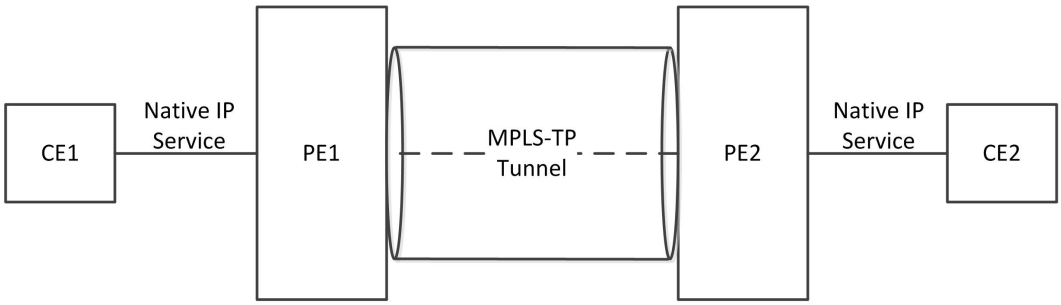

Figure 4-1: Native IP service over MPLS-TP

Reference Documents and Standards

- RFC 3031 — Multi-Protocol Label Switching Architecture
- RFC 5654 — Requirements of an MPLS Transport Profile
- RFC 5586 —MPLS Generic Associated Channel
- RFC 5921 — A Framework for MPLS in Transport Networks
- RFC 5960 — MPLS Transport Profile Data Plane Architecture
- draft-ietf-mpls-tp-identifiers-04 — MPLS-TP Identifiers

- draft-ietf-mpls-tp-cc-cv-rdi-03 — Proactive Connectivity Verification, Continuity Check and Remote Defect indication for MPLS Transport Profile
- draft-ietf-mpls-tp-on-demand-cv-03 — MPLS On-demand Connectivity Verification and Route Tracing
- draft-ietf-mpls-tp-oam-framework-11.txt — Operations, Administration and Maintenance Framework for MPLS-based Transport Network

MPLS-TP Architecture

This chapter provides information about the ZebOS-XP MPLS-TP architecture.The MPLS-TP Infrastructure consists of the following components:

- Tunnel: the top level container for MPLS transport paths from a source to a destination.
- Label Switched Path (LSP): the logical transport mechanism that operates from source to destination. Each tunnel has at least one LSP. Whether a tunnel is unidirectional or bidirectional is determined by its underlying LSPs.

Configurations and paths are stored in tables.

MPLS-TP Tunnels

Three types of tunnels are supported in the MPLS-TP framework: unidirectional tunnel; co-routed bidirectional tunnel; associated bidirectional tunnel; LSPs hold transport paths that comprise of MPLS forwarding information.

Unidirectional Tunnel

Unidirectional MPLS-TP tunnels operate in a single direction from an East node to a West node (source to destination). A Unidirectional Tunnel has at least one unidirectional LSP, which is also referred to as the “forward LSP”

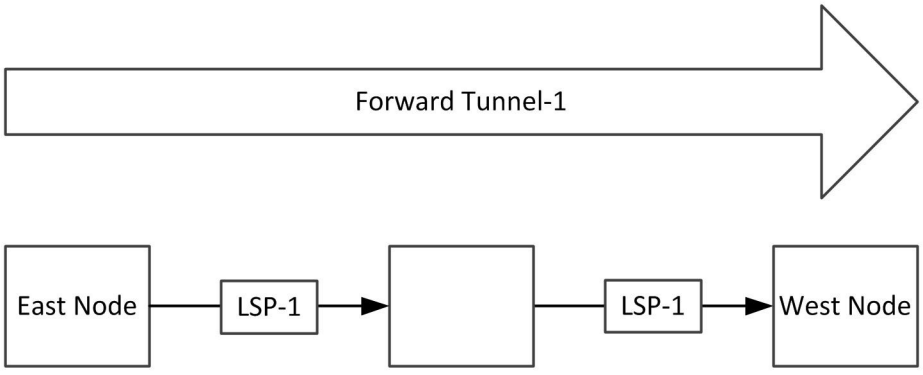

Figure 4-2: Unidirectional Tunnel

Co-routed Tunnel

A co-routed tunnel has at least one bidirectional LSP. A bidirectional LSP has one forward component and one reverse component. Both components traverse the same nodes and links in either direction end to end, from an East node to a West node (source to destination). Each node on the path of a co-routed tunnel maintains the binding between the forward and reverse components for its LSP.

© 2015 IP Infusion Inc. Proprietary

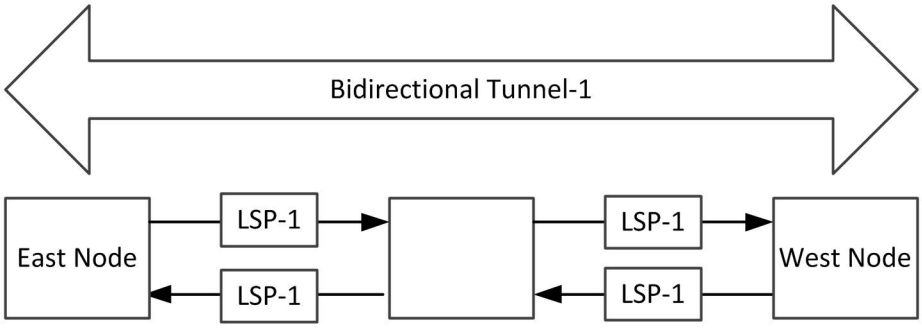

Figure 4-3: Co-routed Bidirectional Tunnel

Associated Tunnel

Associated tunnels support bidirectional data flow with two unidirectional tunnels: one forward tunnel and one reverse tunnel. These tunnels are associated via LSPs.

Each tunnel of an associated-group has at least one unidirectional LSP, which consists of either forward or reverse component, based on the association. The forward to reverse components may traverse either different nodes or different links on the same node, or the same link on the same set of nodes. Two tunnels are “Associated” at the East node, the West node and at all overlapping nodes. On all other non-overlapping nodes, the tunnels are configured as Unidirectional Tunnels.

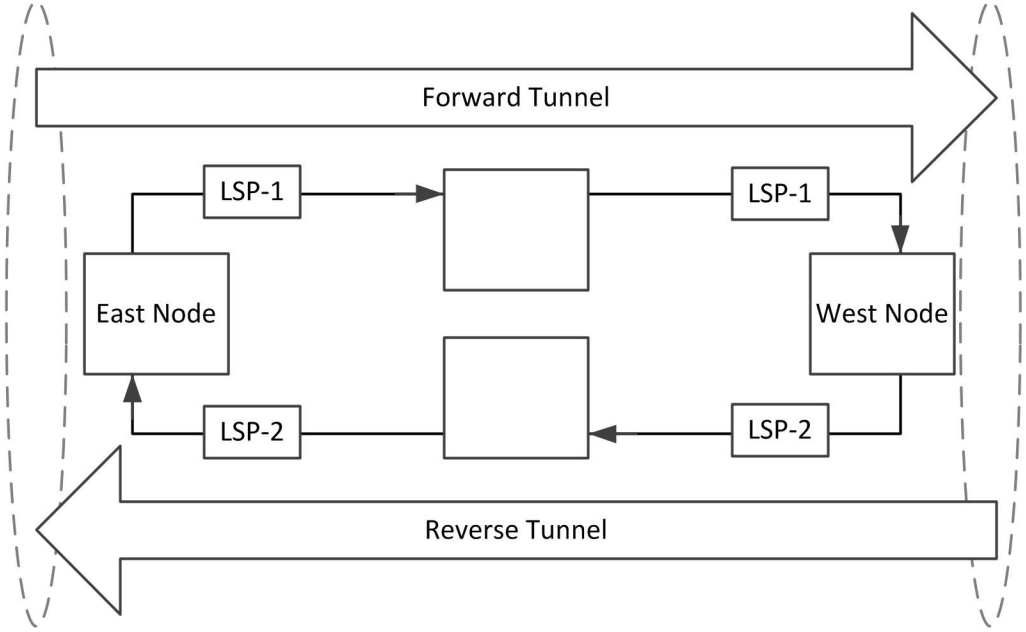

Figure 4-4: Associated Tunnels

MPLS-TP Label Switched Path

Each tunnel can contain one primary Label Switched Path (LSP). (Backup LSPs are supported in future release.) LSPs contain the following information:

FTN Entry

FTN represents FEC-to-NHLFE, a logical table that implements the MPLS architecture defined in RFC 3031: Each FTN entry is used to map incoming traffic to an MPLS LSP. This information is assigned to a NHLFE at the edge of the MPLS cloud. NHLFE represents the Next hop label forwarding, which specifies the MPLS properties for egressing a packet onto LSP. MPLS properties include label, nexthop IP address and outgoing interface

FTN is an entity that is present only on LSP ingress; it is used for PUSHing label information to native packets and tunneling them via LSP. However, in MPLS-TP, incoming packets are not directly mapped to MPLS-TP LSP. Instead, services such as PW map, indirectly route map traffic to the LSP.

For MPLS-TP, FTN is only used to specify NHLFE information for the MPLS-TP LSP at the LSP ingress. FTN implements the forward-component of an LSP on ingress and the reverse component of LSP on egress (for bidirectional LSPs)

ILM Entry and Reverse ILM Entry

ILM represents the Incoming Label Map. ILM is a logical table that is indexed by the incoming interface and label. An ILM entry specifies behavior for processing labeled packets that arrive on MPLS core and egress nodes.

- On core nodes, the behavior is to swap the labels.
- On egress nodes, the behavior is to POP the label and process the native packet.
- For bidirectional LSPs, ILM entries performing label-POP implement the forward component of LSPs on the egress and the reverse components of LSPs on ingress.
- ILM entries performing label-SWAP implement the forward and reverse components of LSPs on transit/core nodes.

Tunnel Modes

Table 4-1 lists the tunnel Modes for MPLS-TP.

Table 4-1: Tunnel Modes

Tunnel Mode	**Ingress**		**Transit**		**Egress**	
	Forward Path	*Reverse Path*	*Forward Path*	*Reverse Path*	*Forward Path*	*Reverse Path*
Unidirectional	NHLFE	N/A	ILM	N/A	ILM	N/A
Co-routed	NHLFE	ILM	ILM	ILM	ILM	NHLFE
Associated	NHLFE	ILM	ILM	ILM	ILM	NHLFE

© 2015 IP Infusion Inc. Proprietary

MPLS-TP Operations, Administration and Management

MPLS-TP utilizes a comprehensive set of Operations, Administration, and Maintenance (OAM) procedures for fault, performance, and protection-switching management that do not rely on the presence of the control plane. On-demand connectivity verification is provided through LSP ping and trace route over the Generic Associated Channel Header Label (GAL/GACh). Proactive connectivity verification support is provided through BFD over GACh.

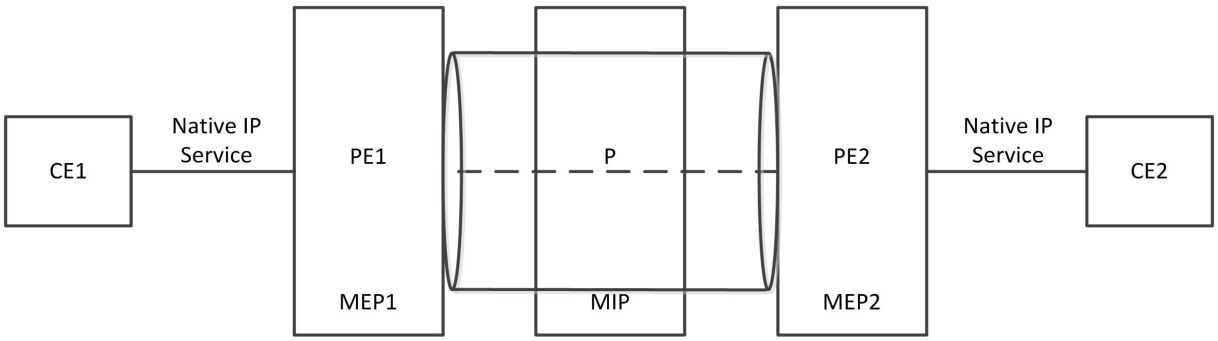

Figure 4-5: OAM Diagram

OAM is comprised of the following components:

Maintenance Entity Group (MEG)

A set of one or more maintenance entities that maintain and monitor a section or a transport path in an OAM domain.

MEG End Point (MEP)

Capable of initiating (source MEP) and terminating (sink MEP) OAM packets for fault management and performance monitoring. MEPs define the boundaries of an ME.

Maintenance Entity (ME)

A portion of a transport path that requires management bounded by two points (MEPs), and the relationship between those points to which maintenance and monitoring operations are applied.

MEG Intermediate Point (MIP)

MIP terminates and processes OAM packets that are sent to this particular MIP and may generate OAM packets in response to received OAM packets; a MIP never generates unsolicited OAM packets. A MIP resides within a MEG between MEPs.

MPLS-TP LSP Ping

The ZebOS-XP MPLS OAMD module supports the LSP-Ping and Trace-Route functions for MPLS-TP. For information about the LSP ping and trace-route commands for MPLS-TP OAM, refer to the *Multi-Protocol Label Switching Command Reference*. Figure 4-6 on page 30 displays the delivery of LSP Ping requests.

© 2015 IP Infusion Inc. Proprietary

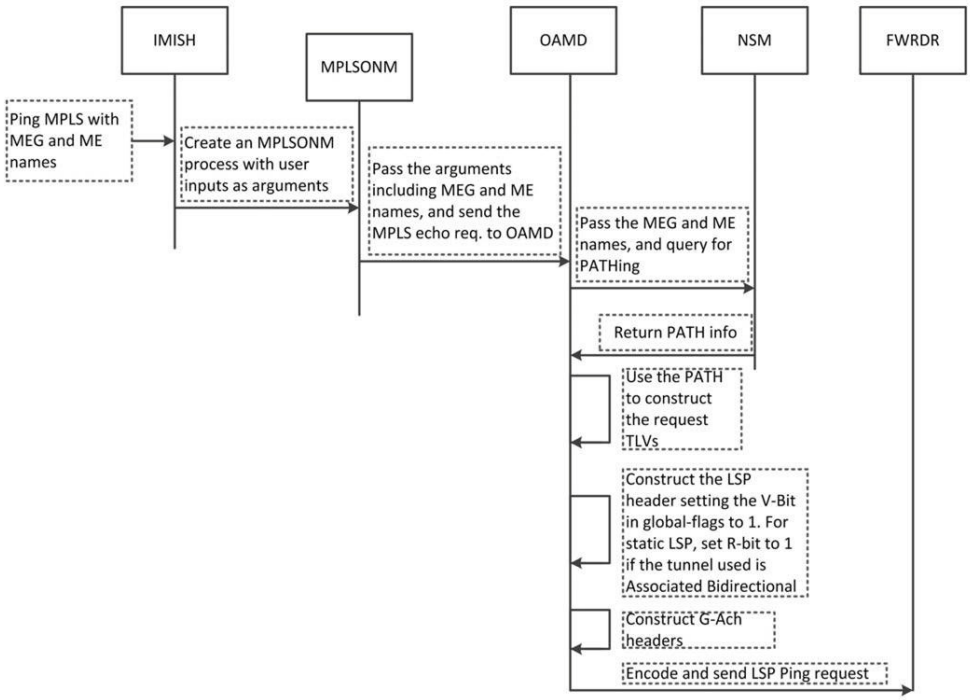

Figure 4-6: Constructing and Sending LSP Ping Request at the Ingress

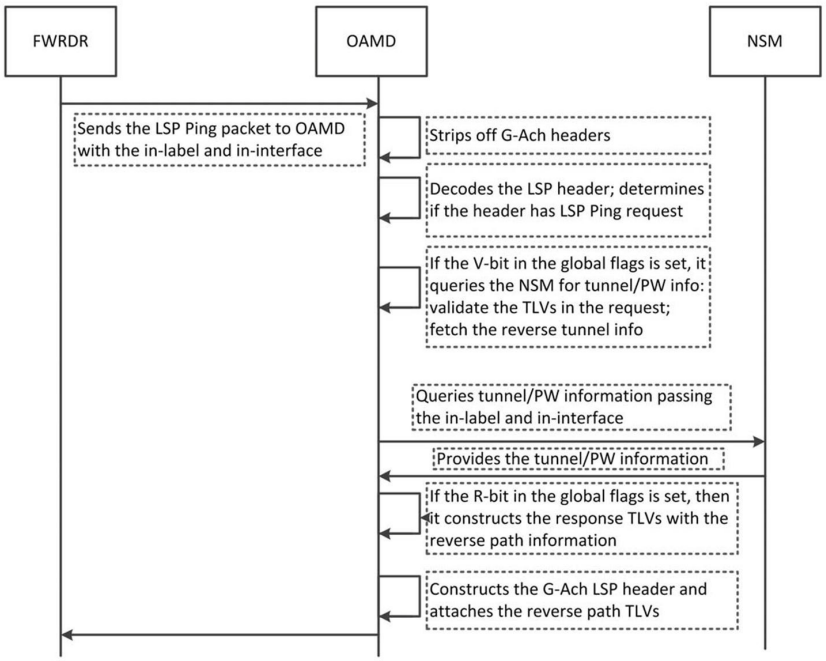

Figure 4-7: Constructing and Sending LSP Ping Request at the Egress

© 2015 IP Infusion Inc. Proprietary

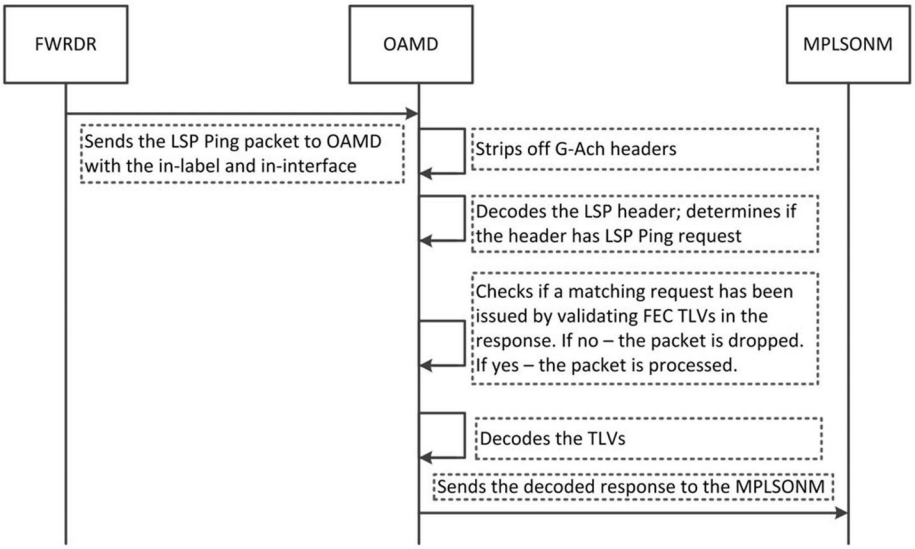

Figure 4-8: Receiving LSP Ping Response at the Ingress

BFD for MPLS-TP LSPs and PWs

The processes involved in BFD management for MPLS-TP LSPs and PWs are described in the sections that follow.

Creating a BFD Session

The following steps provide an overview of how BFD sessions are created.

1. Pointer to the MEG name. and ME name are used as input parameters.
2. The Network Service Module (NSM) sends a Bidirectional Forwarding Detection (BFD) “session add message” to the OAMD module, containing the MEG name and ME name.
3. Upon receiving the session add message, the OAM adds an entry in the new MPLS-TP tree, which is indexed by MEG name and ME name.
4. OAMD creates and sends BFD control packets to the peer.
5. Up to this point, the procedure is same for both the Ingress and Egress nodes; in MPLS-TP, both nodes have MEG and ME configurations for monitoring the transport path.
6. The node receiving the first BFD packet fetches the tunnel or pseudowire (PW) information from NSM based on the input label and input interface, or the virtual circuit (VC) label, respectively.
7. The tunnel or PW information returned by NSM contains the MEG and ME names that are used to find the associated BFD session.
   - When the BFD session goes up/down, the state is communicated to the NSM module.
   - If the tunnel or PW goes down, BFD receives the down notification and stops sending BFD packets. After receiving an UP notification, BFD packet sending is resumed.
   - If the BFD session is made down by administrator on one side, that information is communicated to the peer using the ADMIN down diagnostic code.

© 2015 IP Infusion Inc. Proprietary

Sending BFD Packets

The following steps provide a summary of how BFD packets are delivered.

1. The first BFD packet for the session is constructed with the remote discriminator set to zero.
2. When the BFD session learns the peer's discriminator value, the BFD packets that follow are constructed with the valid remote discriminator field.
3. For BFD Connectivity Verification (CV) packets, the source MEP TLV is included in the BFD packet.
4. The BFD packet is then encapsulated with a G-ACh header.
5. The channel type in the ACH header is filled for BFD Control Channel (CC) and Connectivity Verification (CV) packets.
6. The constructed BFD packet is sent to the MPLS forwarder with the NHLFE index. The forwarder then adds the VC label and/or Tunnel labels to the packet; the packet is then sent out.

Receiving and Processing BFD Packets

The following steps provide is a summary of how BFD packets are received and processed.

1. The OAMD module receives a BFD control packet from the forwarder.
2. The packet is verified.
   - If the packet has a GAL/VC label, then an ACH header should follow that label.
   - The ACH header should have a valid channel type. The channel type could be LSP ping, BFD CC or BFD CV.
   - If the channel type is BFD CV, then the packet should have the source MEP TLV.
3. When packet sanity is verified, the remote discriminator field in the packet is inspected.
4. If the remote discriminator is non-zero, then that value is used to find the BFD session in the BFD Discriminator tree.
   - If the session is found and the packet is CV, then the source MEP in the packet is validated against the configured MEP information.
   - If the MEP TLV is not valid, then the BFD session is moved to the DOWN state.
5. If the session is not found, the packet is dropped.
   - If the remote discriminator is zero, and the packet is received on an MPLS-TP tunnel, the in-label and in-interface values are used to fetch the tunnel information from NSM.
   - If the packet is received on a PW, then the VC label and in-interface values are used to fetch the PW information from NSM, which returns the tunnel or PW information with the MEG and ME names, if they are configured.
6. If the MEG and ME names are not present, the packet is dropped.
7. If the MEG and ME names are present, the MEG and ME names are used to fetch the BFD session from the MPLS-TP BFD tree.
   - If the BFD session is not found, the packet is dropped.
   - If the BFD session is found, the packet is accepted and processed.

© 2015 IP Infusion Inc. Proprietary

Updating a BFD Session

The following steps provide is a summary of updating a BFD session.

1. NSM receives BFD configuration changes for MPLS-TP for a MEG name and ME name.
2. NSM forms the BFD session addition message and sends it to the OAMD.
3. After receiving the session message, the OAMD finds the BFD session and sets the session state to ADMIN down.
4. The ADMIN down state is communicated to the peer.
5. After receiving the ADMIN down state, the peer sets its BFD session to down.
6. Modifications are made to the session and the session is then set to UP.
7. The session state change information is then sent to the peer.

Deleting a BFD Session

The following steps provide a summary of deleting a BFD session.

1. NSM receives the BFD deconfiguration for a MEG name and ME name.
2. NSM forms the BFD session deletion message and sends it to the OAMD.
3. After receiving the delete message, the OAMD finds the BFD session and sets the session state to ADMIN down.
4. The ADMIN down state is communicated to the peer and the session delete timer is started.
5. After receiving the ADMIN down state, the peer sets its BFD session to DOWN.
6. When the timer expires, the session entry from the MPLS-TP BFD and BFD Discriminator tree is deleted.

The following illustrations depict the construction, delivery and processing of BFD packets.

© 2015 IP Infusion Inc. Proprietary

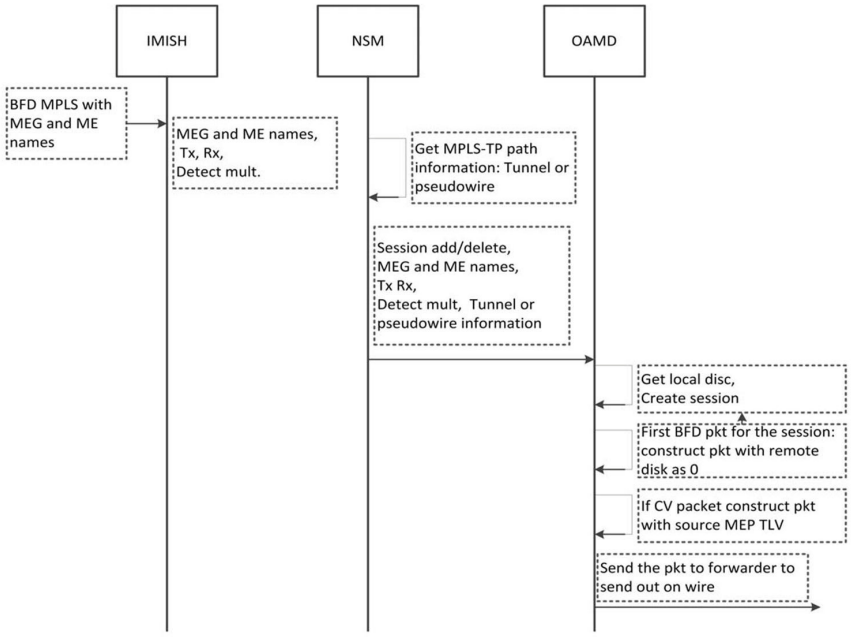

Figure 4-9: Constructing BFD Packets for Session, Ingress or Egress

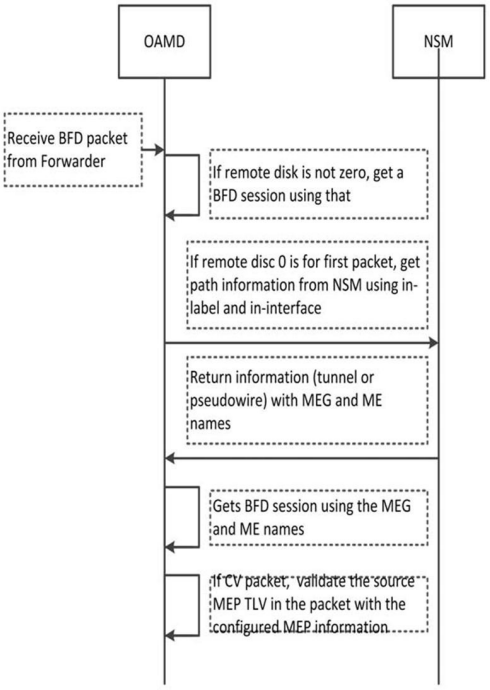

Figure 4-10: Received and Processing BFD Packets for Session, Ingress or Egress

Data Link OAM Sessions

Following is an overview of how OAM data link sessions are set up.Datalink OAM sessions are used to monitor the link status of a point-to-point MPLS provider interface

1. The Datalink OAM BFD session information is stored in an AVL Tree with a key as an interface index.

 © 2015 IP Infusion Inc. Proprietary

- BFD runs in active mode on both ends for datalink OAM sessions.
- OAMD also constructs and adds the G-ACh headers, and sends the packet to the forwarder.

2. The forwarder forwards the OAM packet to the Egress side.
   - In the Egress node, if the label is a GAL label, the forwarder classifies the packet as data link OAM packet.

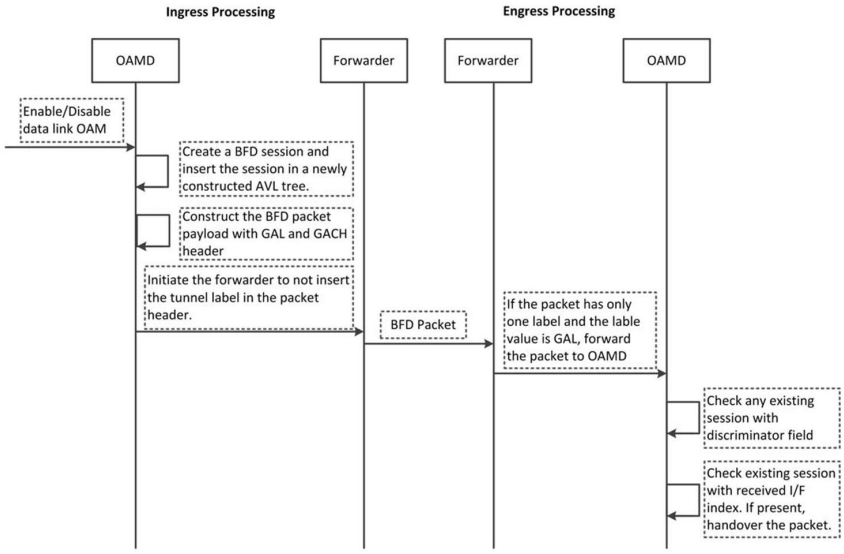

Figure 4-11: MPLS TP Datalink OAM Support, Ingress or Egress

MPLS-TP Tables

This section describes the MPLS TP tables and routing information bases.

MPSL-TP Tunnel Tables

Individual tables support the look-up process. When a tunnel is configured, it is stored in the appropriate table after determining the role of the node for that tunnel: Ingress, Transit or Egress. Roles are determined by comparing the node's tuple (GlobalID, NodeID) to the tunnel's configured end-points.

Note: The data structures are defined in the file nsm_mpls.h.

Ingress Tunnel Table

- Stores all tunnels originating at the current node

Transit Tunnel Table

- Stores all tunnels transiting through the current node

Egress Tunnel Table

- Stores all tunnels terminating on the current node

Tunnel Hash Table

- Identifies tables with a Tunnel Name
- Maintains Tunnel Name to Tunnel Mapping

- Tunnel Name is used to map services such as PW and IPv4 map routes to an MPLS-TP Tunnel

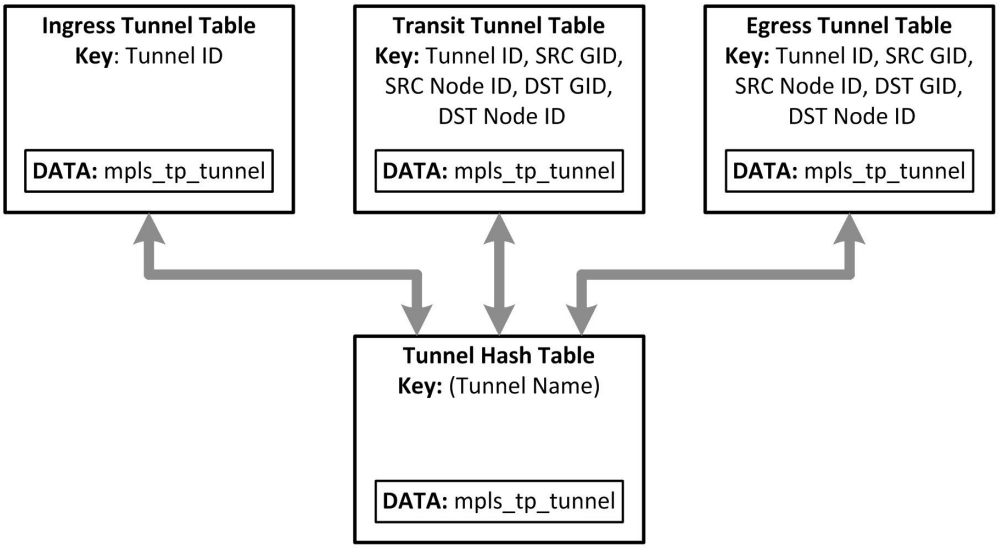

Figure 4-12: MPLS-TP Tunnel Tables

MPLS-TP Routing Information Base

The MPLS-TP Routing Information Base (RIB) infrastructure defines forward / reverse path forwarding entities.

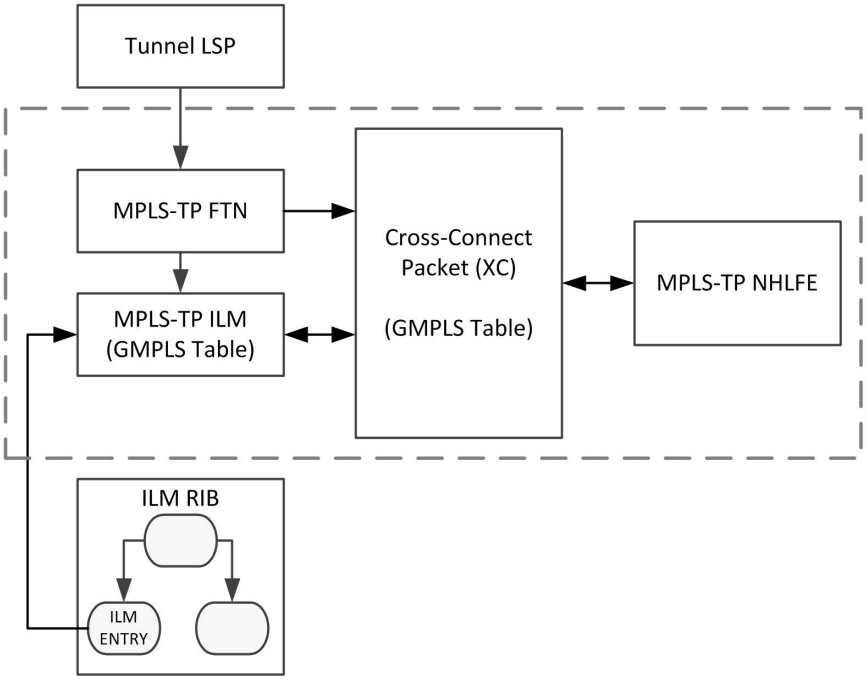

Figure 4-13: MPLS-TP RIB Tables

MPLS-TP Tunnels—RIB Map

This section describes the MPLS-TP tunnel mappings to the MPLS-TP RIB.

© 2015 IP Infusion Inc. Proprietary

Unidirectional

.Each tunnel has an LSP that holds either an FTN or an ILM entry at the tunnel end-points. Two unidirectional tunnels are illustrated in the following figure; Tunnel 1 originates in this node and Tunnel 2 terminates in this node.

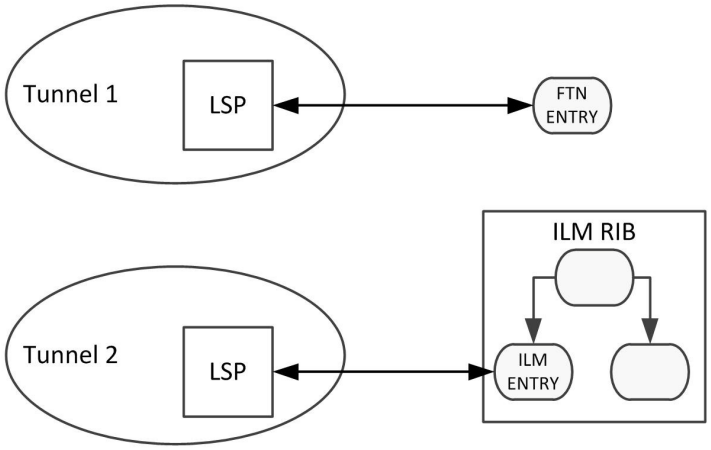

Figure 4-14: RIB Map—Unidirectional Tunnels

Associated Bidirectional

Associated Bidirectional tunnels consist of two tunnels: forward and reverse. Forward and reverse tunnels may traverse different sets of nodes or links or both. Each tunnel has an LSP that holds either the forward or the reverse component depending on semantics of this tunnel with the associated binding. The following figure shows a high level design for a set of tunnels forming an Associated Bidirectional tunnel.

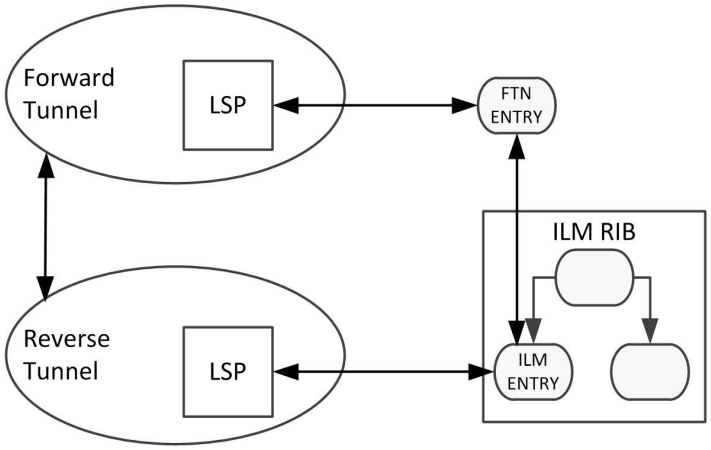

Figure 4-15: Associated Bidirectional Tunnel

Co-routed Bidirectional

Co-routed tunnels are bidirectional tunnels with their forward and reverse paths sharing fate on the same set of nodes and links end-to-end. Each tunnel has an LSP that holds a forward component and a reverse component. At the endpoints, the forward component is the FTN entry and the reverse component is the ILM entry. The following figure illustrates a co-routed bidirectional tunnel.

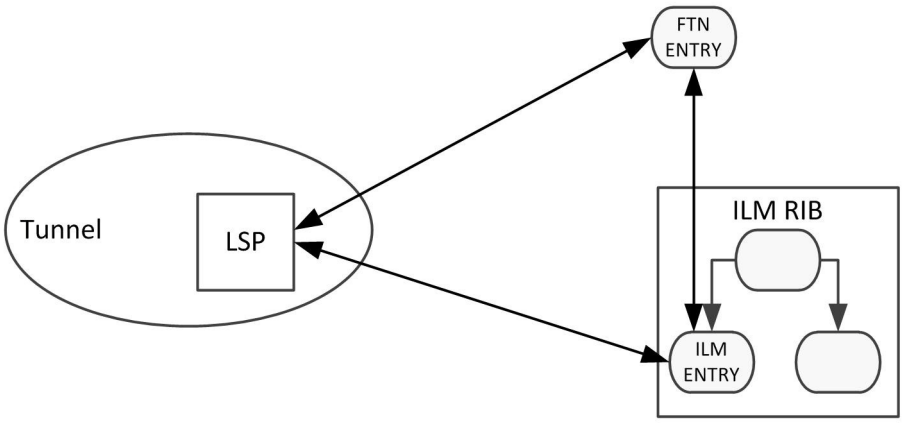

Figure 4-16: Co-routed Bidirectional Tunnel

© 2015 IP Infusion Inc. Proprietary

CHAPTER 5 Layer 2 Virtual Private Network

L2VPN Provisioning with BGP and LDP

A network administrator gives the `protocol` command in MPLS to configure and manage VC/VPLS with BGP auto-discovery and LDP signaling.

Once a VC/VPLS is configured, NSM forwards a message to BGP which creates of the Layer 2 Virtual Private Network (L2VPN) instances, peer discovery for the L2VPN address family using NLRI messages. BGP also interacts with NSM to inform the auto-discovered peers.

When BGP auto-discovery is complete, BGP sends peers information to NSM to trigger signaling through LDP. BGP auto-discovery enables PE routers to discover other PE routers that are part of the same VC/VPLS domain. Signaling also sets up and tears down pseudowires with other PE routers.

BGP Auto-Discovery

Discovery refers to the process of finding all PEs that participate in a VC/VPLS instance. This allows each PE configuration to identity the VC/VPLS instance established on the PE, but not identity every other PE in the VC/VPLS instance.

Route target is used as a demultiplexer for discovering the PEs that belongs to a VC/VPLS instance.

When BGP distributes end point provisioning information in an update message to all of its neighbors, the information is used to configure a pseudowire to support L2VPN-based services.

LDP Signaling

Signaling is, at its most basic, the set up and tear down of pseudowires. Once BGP auto-discovery is accomplished, each pair of PEs in a VC/VPLS must be able to set up and tear down pseudowires with each other. Signaling is also used to send characteristics of the pseudowires that a PE sets up for a VC/VPLS. NSM interacts with LDP to signal the VC/VPLS information to the discovered PEs to set up the pseudowire.

BGP Address Family

When a network administrator gives the `address-family l2vpn vpls` commnd, the mode changes from the BGP router mode to BGP VPN mode. In this mode, a network administrator configures the neighbor and the network for the VC/VPLS instance. Once configuration is done, BGP updates its structure with the address family information.

The following figure illustrates the process of VPLS configuration with BGP auto-discovery and signaling.

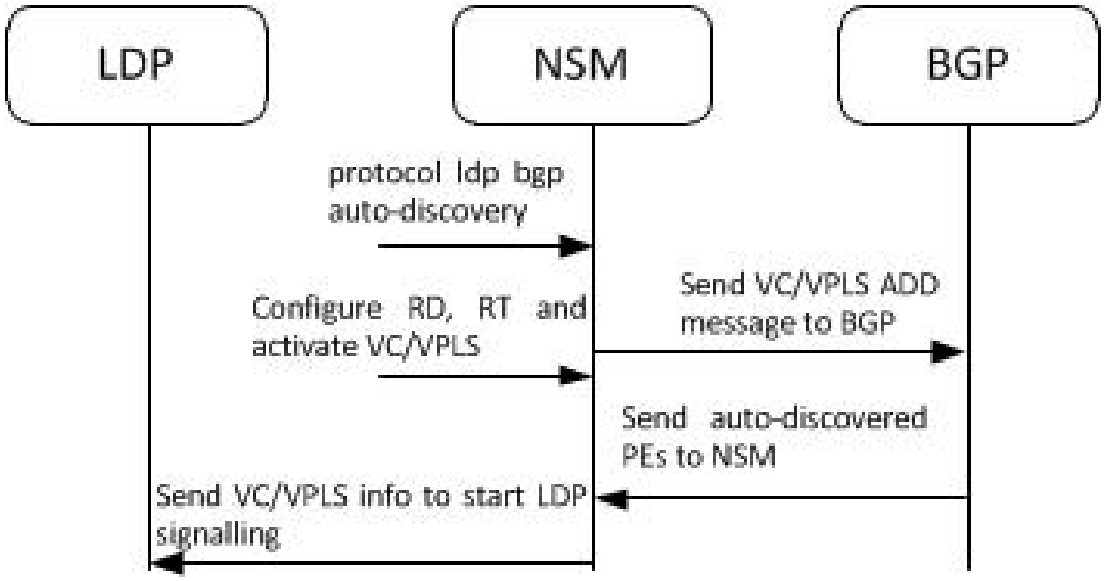

Figure 5-1: L2VPN with BGP auto-discovery and LDP signaling

Command API

l2vpn_vpws_create

This function creates a Layer 2 Virtual Circuit.

This function is called by the `l2vpn-vpws` command.

Syntax

```
s_int32_t
l2vpn_vpws_create (struct nsm_master *nm, char *vc_name, int *ret )
```

Input Parameters

`nm`	Specify the NSM Master

Output Parameters

None

Return Values

NSM_SUCCESS/NSM_FAILURE

l2vpn_create_vpls_peer

This function creates a mesh peer for an VPLS instance.

This function is called by the `neighbor A.B.C.D` command.

Syntax

```
s_int32_t
```


© 2015 IP Infusion Inc. Proprietary

```
l2vpn_create_vpls_peer (struct nsm_master *nm, struct nsm_vpls *vpls,
struct addr_in *addr, u_char fec_type_vc)
```

Input Parameters

`nm`	Specify the NSM Master
`vpls`	Specify the VPLS instance
`addr`	Specify the mesh peer address
`fec_type_vc`	Specify the FEC type

Output Parameters

None

Return Values

NULL – When VPLS mesh peer creation fails

© 2015 IP Infusion Inc. Proprietary

CHAPTER 6 MPLS-TP OAM Modules

MPLS-TP OAM (Operations, Administration, and Maintenance) plays a significant role in providing methods for fault management and performance monitoring in both the transport and the service layers thereby improving their ability to support services with guaranteed and strict Service Level Agreements (SLAs) while reducing their operational costs. OAM for MPLS-based transport networks consists of a comprehensive set of fault indication and performance monitoring capabilities.

Fault Management

Proper operation of a transport network depends on the ability to quickly identify faults and focus attention on the root cause of the disruption. When a fault occurs in a server layer, Fault Management (FM) OAM messages are sent to clients of that server so that alarms, which otherwise would be generated by the subsequent disruption of the clients, may be suppressed. This prevents a storm of alarms and allows operations to focus on the actual faulty elements of the network. Two broad classes of service disruption are supported:

Faults

A fault is an inability of a function to perform a required action. This does not include an inability due to preventive maintenance, lack of external resources or planned actions.

Locks

A lock is an administrative state in which it is expected that only test traffic, if any, and an OAM message can be sent on an LSP.

Server Failure Notification

A server-failure occurs when a fault condition or conditions have persisted long enough to consider the required service function of the server sub-layer to have terminated. In the case of a protected server, this means that the working facilities and any protection facilities have all suffered faults of operator-configured durations.

Alarm Indicator Signaling

The MPLS Alarm Indication Signal (AIS) message is generated in response to detecting faults in the server layer. The AIS message is sent as soon as the condition is detected. The primary purpose of the AIS message is to suppress alarms in the network layer above the level at which the fault occurred.

Link Down Indication

The Link Down Indication (LDI) is communicated by setting the L-Flag in the AIS message to 1. A node sets the L-Flag in response to detecting a failure in the server layer. The receipt of an AIS message with the L-Flag set MAY be treated as the equivalent of loss of continuity (LOC) at the client layer. The setting of the L-flag can be predetermined based on the protection state. If the server layer is unprotected or the server layer is protected but only the active path is available, the node sends an AIS FM message with the L-flag set upon detecting a LOC condition.

Lock Report

MPLS Lock Report (LKR) messages are generated when a server layer entity has been administratively locked. It communicates the locked condition to the client-layer entities. LKR messages suppress alarms in the network layer above the level at which administrative locks occurs, which helps differentiate a lock condition from a fault condition.

Propagation of Messages

MPLS-TP supports a hierarchy of LSPs. When the client maintenance end point (MEP) of an LSP that is also acting as a server layer receives FM indications, the following rules apply.

- If the Continuity Check (CC) function is disabled for the server LSP, a node generates AIS messages for any clients when either an AIS or a LKR notification is received.
- An L-Flag is not automatically propagated; L-flags are not set until a server failure is declared.

Architecture

MEG main card and line card architectures allow fault management to be logically split into two parts: control-card and line-card-driven modules. The main-card module supports configuration, reconfiguration, removal of configurations and maintenance of MEG and OAM functions. The line-card module runs the timers, creates and sends, and receives and processes OAM packets.This architecture is crucial for time sensitive applications like CC-CV or protection switching, where detecting faults and protecting the traffic via backup path should be achieved as quickly as possible.

When a MEG is configured and the operational status is UP, the main module creates a MEG entry in the LINE module via a distributed API layer. The distributed API layer acts as an abstraction layer between MAIN and LINE modules. Any updates in the main or line modules are communicated via this abstraction layer. Once the MEG is created in the line module, it can start any OAM functions configured, including CC-CV, FM, Lock Instruct (LI) Loopback (LB), or protection switching coordination.

Maintenance Entity Server-Client Relationship

A server-client relationship is supported in MPLS-TP fault-management, wherein notification of a fault detected by a server layer MEP is reported to any client layer MEPs. An MPLS-TP LSP uses the MPLS-TP provider interfaces (datalink layers) to carry traffic across the LSP. The datalink layer that provides the service becomes the server layer and the LSP becomes the client.

The server-client relationship is built dynamically by identifying the LSPs that use a datalink provider interface. When a MEG is configured for a datalink provider interface, it becomes the server. Subsequently, when a new MEG is created for an MPLS-TP LSP that makes use of this interface as the incoming interface, it becomes the client layer. The ME for a datalink provider interface maintains a list of client MEs, and new MEs are added to the list as they are created.

Fault Management Processing

These sections describe the processes involved in MPLS-TP fault management,

Entering a Fault State

An MEP (maintenance end point) enters a fault state when it is locked or the transport path is declared as DOWN. An MEP can also enter a fault state when it receives an FM notification from the server layer MEP. If an MEP is in the fault

© 2015 IP Infusion Inc. Proprietary

state, and it is a server layer with associated client layers, it starts sending FM notifications to the associated client layers. It continues to periodically send FM notifications until it exits from the fault state.

Exiting a Fault State

The fault state is can be exited when an FM message indicating the removal of the fault is received. When an MEP exits from a fault state, it stops sending FM notifications to any client MEPs and sends an FM message to indicate the removal of the fault. An MEP can also exit from a fault state when the configured FM refresh timer expires.

Creating and Sending Notifications

When a fault is detected, the OAM daemon (OAMD) determines whether the MEG has any associated clients. If so, the OAMD begins notification with FM messages. The OAMD creates FM messages based on the type of fault detected.

Notification for Lock Condition

When the MEP is locked, a LKR message is generated and sent to clients. LKR messages indicate that the MEP is locked for maintenance purposes. The version number of the FM message is set to 1, the reserved fields are set to 0 and the message type is set to 1. The L-flag is not set for LKR messages, since there is no link down condition.

The FM message includes the interface identifier and global identifier TLVs. The global identifier is the node's global ID, and the interface identifier is the server's MEP interface ID, which is also the incoming interface of the client LSP or PW. The new FM message is encapsulated in a generic associated channel (G-ACh) header with a channel type of 0x0058. If the client layer is an LSP, then the G-ACh header is also encapsulated with a GAL label. The encapsulated FM message is sent to the MPLS forwarder along with tunnel or virtual circuit information. The forwarder adds a VC label, or a VC label plus a tunnel label, then sends the message to the clients. Messages are retransmitted twice with an interval of one second.

The message timer is started with the configured refresh timer to periodically send the message to all clients. When the ME is declared UP. either by receiving an UNLOCK for MEP or a notification that the PATH is UP, the FM message timer is stopped. The R-flag is set in the FM message and it is sent to all clients to indicate that the fault condition is cleared. This message is also retransmitted twice with an interval of one second.

Notification of Path Down Condition

If a PATH DOWN condition is in effect, OAMD generates an AIS FM message, which indicates that the server layer is suffering an actual fault. The version number of the FM message is set to 1, the reserved fields are set to 0 and the message type is set to 1 for AIS. When the message is of the type AIS and the MEG is not protected, the L-flag is set to notify clients of a link down state.

The FM message includes the Interface Identifier and Global Identifier TLVs. The Global Identifier is the node's global ID, and the Interface Identifier is the server's MEP Interface ID, which is also the incoming interface of the client LSP or PW. The new FM message is encapsulated in a GACh header with a channel type of 0x0058. If the client layer is an LSP, then the G-ACh header is also encapsulated with a GAL label. The encapsulated FM message is sent to the MPLS forwarder along with tunnel or virtual circuit information. The forwarder adds a VC label, or a VC label plus a tunnel label, then sends the message to the clients. Messages are retransmitted twice with an interval of one second.

The message timer is started with the configured refresh timer to periodically send the message to all clients. When the ME is declared UP. either by receiving an UNLOCK for MEP or a notification that the PATH is UP, then the FM message timer is stopped. The R-flag is set in the FM message and it is sent to all clients to indicate that the fault condition is cleared. This message is also retransmitted twice with an interval of one second.

Receiving and Processing Notifications

The actions involved in receiving and processing of notifications are described below.

- The OAMD receives FM messages from the forwarder. If the FM message is received from a tunnel then the in-label and in-interface ID pair are used to find the corresponding MEG for the message.
- If the FM message received is for a VC then the VC label is used to find the MEG. If no MEG is configured the FM message is ignored; if a MEG is configured, then packet sanity is verified.
- MEG checks to determine whether the FM message indicates a new fault or is sending refreshed information for a fault already reported. In the case of a refresh message, the refresh timer is restarted. If the message is reporting a new fault, the fault indicated is entered for the MEG. When the message contains Interface or GLOBAL ID TLVs, the TLVs are stored in the MEG.
- A refresh timer starts for the new fault with the timer value as 3.5 times the refresh timer value in the FM message.
- When CC is enabled for the MEG, the CC session is declared as DOWN. If the MEP has any associated client layers, fault notification is generated and sent to those MEPs.
- If the FM message has the R-bit set, the MEG is checked to see whether the reported fault exists in the MEG. If not, the FM message is ignored. If a fault does exist, the TLVs are verified with the stored TLVs. If they match, the fault exits and the refresh timer stops. If the ME is UP and the ME has any associated client layers that MEP is sending FM messages with no R-bit set, the FM message is sent with the R-bit set to indicate that the fault is cleared.

Packet Loss and Delay Measurement

Many provider service level agreements (SLAs) depend on the ability to measure and monitor performance metrics for packet loss and one-way and two-way delay measurements, and other metrics, such as delay variation and channel throughput. These measurement capabilities provide operators with greater visibility into the performance characteristics of their networks, thereby facilitating planning, troubleshooting, and network performance evaluation. The packet loss and delay measurement mechanisms implemented in ZebOS-XP enable the efficient and accurate measurement of these performance metrics in MPLS and MPLS-TP networks.

Overview

The following figure depicts a bidirectional channel between two nodes, A and B.

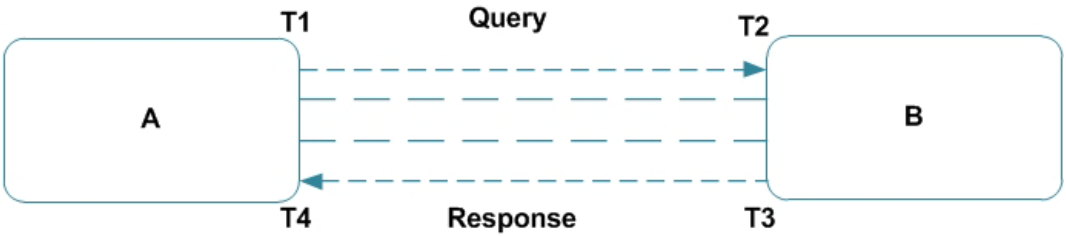

Figure 6-1: Bidirectional Channel Reference

The illustration depicts four temporal reference points, T1 through T4, associated with a measurement operation that takes place at A. The operation consists of node A sending a query message to node B, and B sending back a response. Each reference point indicates the point in time at which either the query or the response message is transmitted or received over the channel.

Node A can arrange to measure any packet loss over the channel in the forward and reverse directions by sending Loss Measurement (LM) query messages to node B, each of which contains the count of packets transmitted prior to the time T1 over the channel to B. When the message reaches node B, it appends two values and reflects the message back to node A: The two values are the count of packets received prior to time T2 over the channel from A the count of packets transmitted prior to time T3 over the channel to A. When the response reaches node A, it appends a fourth value: the count of packets received prior to time T4 over the channel from B. These four counter values enable node A to compute the desired loss statistics.

© 2015 IP Infusion Inc. Proprietary

To measure at node A the delay over the channel to node B, a Delay Measurement (DM) query message is sent from A to B containing a timestamp recording the instant at which it is transmitted; this is T1. When the message reaches B, a timestamp is added recording the instant at which it is received; this is T2. The message can now be reflected from B to A, with B adding its transmit timestamp, T3, and A adding its receive timestamp, T4. The four timestamps enable node A to compute the one-way delay in each direction, as well as the two-way delay for the channel. The one-way delay computations require that the clocks of node A and node B are synchronized.

Features

The Loss Measurement (LM) and Delay Measurement (DM) modules are intended to be simple and to support efficient hardware processing. Both LM and DM operate over an MPLS G-ACh and supports measurement of loss, delay, and related metrics over LSPs, PW, and MPLS sections (links). One-way delay-measurements are associated with forward or reverse paths, while two-way delay measurement is employed in bidirectional channels.

The LM and DM modules use a simple query-and-response model for bidirectional measurement that allows a single node, the querier, to measure the loss or delay in both directions. LM and DM use query messages for unidirectional loss and delay measurement. The do no require that the transmitting and receiving interfaces be the same when performing bidirectional measurements. The DM module is stateless, while the LM module is almost stateless. In LM, a loss is computed as a delta between successive messages, and thus the data associated with the last message received is retained. The LM module can perform two distinct kinds of loss measurement: inferred measurement, where test messages are used, and direct measurement, where data packets are used for loss measurement. It supports measurement in terms of both packet counts and octet counts. The LM module supports both 32-bit and 64-bit counters, and can be used to measure channel throughput as well as packet loss.

The DM module supports multiple timestamp formats, providing a simple means for two endpoints of a bidirectional connection to agree on a preferred format.

Process Flow

To configure an on-demand session, the commands provided require you to enter a MEG name, a ME name, and an interval desired between packets as options. For a proactive session, only the interval between packets need to be configured. All commands necessary to configure LM and DM are supported in the ZebOS-XP Integrated Management Interface Shell (IMISH). IMISH uses the MPLS ONM module to manage on-demand sessions.

When configuration is complete, the OAMD validates that the MEG and ME given in the commands are configured; if not, an error is returned. Next, validations are performed on the tunnel, PW or datalink section associated with the MEG and ME to ensure that they are up. An error is returned if the operational status of any required entity is down. An error is also returned when any attempt is made to initiate an LM/DM session from an intermediate node. After all arguments are validated, the OAMD invokes the HAL API to start the session. On-demand sessions are maintained for the duration configured, and proactive sessions are maintained until a command is issued to terminate them.

OAMD uses the HAL API to retrieve LM/DM statistics and send them to MPLSONM for display. HAL is responsible for constructing the LM and DM packet payload, attaching GACh headers and the MPLS label stack. HAL has the forward and reverse path information and uses it to forward request and response packets. HAL updates the counters in an LM message and the timestamps in a DM message, and computes the total transmit loss and receive loss for an LM session and passes the information to the OAMD. The HAL also computes the one-way delay associated with a forward and a reverse path, or the two-way and round-trip delays associated with bidirectional channels.

Lock and Loopback

Lock and loopback are two additional OAM functions that are useful in a transport network.

Lock Function

The lock function enables an operator to lock a transport path so that it does not carry client traffic, but can continue to carry OAM messages and may carry test traffic.

Locking a Transport Path

When a MEP receives a Lock command from a network management system or through some other management process, it must take the transport path out of service. That is, it must stop injecting or forwarding traffic onto the LSP, PW, or bidirectional section that has been locked. When locking a transport path, the management system or process is required to send a lock command to both ends of the transport path. If rapid coordination of lock state is to be achieved. then as soon as the transport path has been locked, the MEP must send a lock instruct (LI) message targeting the MEP at the other end of the locked transport path until it is unlocked by the management process.

Unlocking a Transport Path

Unlock is used to request that a MEP bring the previously locked transport path back into service. When a MEP receives an Unlock command from a management process, it must cease sending LI messages. If the MEP is still receiving LI messages, the transport path must remain out of service. To unlock a transport path, the management process has to send an Unlock command to the MEPs at both ends.

Creating and Sending Lock Instruct Packet

The version field is set to 1 and the reserved fields are set to 0 in the LI message. The configured refresh timer value is set in the refresh timer field. The Tunnel or PW source MEP TLV is included in the LI message based on the path type. The LI message is encapsulated with the GAL label and the ACH header with the channel type set to 0x0026. The constructed LI packet is sent to the forwarder with the Tunnel NHLFE index or PW VC ID included. The forwarder encapsulates the packet with a VC label or a VC label plus a tunnel label and sends the packet out.

Lock Instruct Packet Reception and Handling

When the forwarder receives the packet, it sends it on to the OAMD, which uses the VS label (if received on a PW) or in-label and in-interface (if received on a tunnel) to identify the MEG. If the MEG is not found, the packet is discarded, otherwise, packet sanity validation is performed, including channel type. An LI packet should have a valid source MEP TLV, otherwise, the packet is dropped.

Loopback Function

The loopback function is used to test the integrity of a transport path from an MEP to any other node in the same MEG This is achieved by setting the target node into loopback mode, and transmitting a pattern of test data from the MEP. he target node loops all received data back toward the originator, and the MEP extracts the test data and compares it with what it sent. Loopback is a function that enables a receiving MEP or MIP to return traffic to the sending MEP when in the loopback state. This state corresponds to the situation where, at a given node, a forwarding plane loop is configured, and the incoming direction of a transport path is cross-connected to the outgoing reverse direction. Therefore, except in the case of early TTL expiration, traffic sent by the source will be received by that source.

Putting a Node Into Loopback Mode

To put a MEP or MIP in loopback mode, you must configure it. When the OAMD receives the `loopback` command it validates the ME entry. If the `is_loopback_enabled` flag is already set to TRUE, the command is ignored. If the flag is set to FALSE, then the MP type is checked to determine whether it is MIP or MEP. If it is a MEP, the MEP must be locked before applying the loopback. If the MEP is not locked, an error message is returned.The `is_loopback_enabled` flag is then set to TRUE and a message is sent to NSM with the MEG name, ME name and the path identifier. OAMD calls the Loopback API to put the node in loopback mode.

© 2015 IP Infusion Inc. Proprietary

Removing a Node from Loopback Mode

To remove a MEP or MIP from loopback mode, you must explicitly update it. When OAMD receives the CLI command it validates whether the ME entry has the `is_loopback_enabled` flag is set to FALSE, the command is ignored. If the flag is set to TRUE, it is then set to FALSE and a message is sent to NSM with the MEG name, ME name and the path identifier. OAMD will calls Remove Loopback API to remove the node from loopback mode.

© 2015 IP Infusion Inc. Proprietary

CHAPTER 7 MPLS-TP OAM Y-1731 Support

This chapter discusses Y-1731 support for the MPLS-TP OAM feature.

System Architecture

The Y-1731 distributed architecture allows to logically the splitting of the OAM module into two parts: control card and line card modules. The control card OAM allows you to configure and maintaining MEG and OAM functions. The line card module actually runs the timers, as well as create, send receive, and process OAM packets. Both functions are crucial for time sensitive applications, such as CC-CV and protection switching, where detecting faults and protecting the traffic via backup path must be achieved within 50 milliseconds.

System Features

The Y-1731-based MPLS-TP OAM provides the following features:

- OAM PDUs and procedures that meet the transport networks requirements for OAM
- An encapsulation mechanisms to carry OAM PDUs within Ethernet frames, which provides Ethernet OAM capabilities in Ethernet networks
- Although Y.1731 supports Ethernet OAM, the definition of OAM PDUs and procedures is technology independent and can be used in other packet technologies (e.g., MPLS-TP), provided that the technology-specific. encapsulation is defined.

Library

Since the various aspects of Y.1731 OAM are technology independent, it is more desirable to define the technology independent part of Y.1731 in library. This includes the following.

- All the FSMs defined in OAM functions and mechanisms for Ethernet based networks IEEE P802.1ag/D8.1 Virtual Bridged Local Area Network.
- OAM PDU generation that do not include generation of TLV.
- Includes portion of MEP, MIP, and RMEP data structures.
- Appropriate callbacks and the callback registration function is provided to perform the technology specific actions. This includes:
    - Encapsulation and De-capsulation mechanism to carry the OAM PDU.
    - Placeholder for MEP, MIP, RMEP data structures.
    - Construction and processing of OAM TLVs.

Control Card OAMD

The control card OAMD provides infrastructure to configure and maintain IETF maintenance entities such as MEG, ME and it association with transport paths. This helps maintain the ITU maintenance entities (MEG, MEP, and RMEP) and IETF maintenance entities, as well. The existing MEG table in control card will be used for storing both the IETF and ITU MEG entries. The meg_type field in MEG entry will determine whether the MEG is IETF MEG / ITU MEG.

In addition, the control card OAMD provides infrastructure to configure and manage the OAM maintenance entities such as MEG, MEP, and RMEP, along with an association with transport path information. The structure to store ITU MEG entities, such as meg, mep, rmep, and mip is defined inside library. The data structure that holds this information is defined in OAMD. The existing MEG tables, including mplstp_meg_name_tree, mplstp_meg_index_tree, mplstp_meg_me_index_tree, mplstp_provider_if_tree, and mplstp_provider_if_client_tree, are used to store both the IETF and ITU MEG entries. Again, the meg_type field in MEG entry determines whether the MEG is IETF MEG / ITU MEG.

The ME/MEP information is generic to all transport path technologies inside a library. The ME/MEP information is very specific to the transport path technology as a void pointer into it. This provides the flexibility of reducing the effort to enhance OAM function based on Y.1731 to different transport path technologies.

Control Card NSM

Network service Manager (NSM) manages the information related to MPLS-TP transport path, such as MPLS-TP tunnel, PW and datalink. NSM shares this information to other modules of ZebOS-XP using asynchronous sockets created at startup. It handles the creation and maintenance of these transport paths. Whenever a new MEG is associated with the transport path, OAMD sends a message to NSM to get the transport path information. NSM responds to this query by sending back the transport path information to OAMD.

Control Card IMISH

- IMISH Module in ZebOS-XP handles the following additional responsibilities:-
- Provide commands for triggering all the on-demand y.1731 MPLS TP OAM functions like LBM, TST, LM, 1DM, 2DM etc.
- Provides commands for enabling all the proactive y.1731 MPLS TP OAM functions like CC, LM, AIS, LCK functions etc.

Control Card MPLSONM

The MPLSONM process handles requests from the user for all on-demand OAM functions. MPLSONM sends the on-demand OAM request message to the OAMD module for processing and then waits for a reply from the OAMD module. MPLSONM module is an existing module in ZebOS-XP. It parses the on-demand LBM, on-demand test, loss measurement and delay measurement received from the user. It also establishes IPC communication with the OAMD module using the BFD client APIs to send/receive the above mentioned command's request and response.

Line Card

The line card contains the major part of the Y.1731 based MPLS TP OAM. The line card OAM provides infrastructure to maintain MEG-related information to perform the various IETF MPLS TP OAM tasks. An existing MEG table in the line card stores both the IETF and ITU MEG entries. The type field in MEG entry determines whether the MEG is IETF MEG or ITU MEG.

Similar to control card, the structure to store ITU MEG entities is defined inside library. The data structure that holds the information that is defined in OAMD.

NSM module creates the MPLS TP tunnel and VC using the identifiers defined in draft-ietf-mpls-tp-itu-t-identifiers-03. An mpls-tp tunnel creates the source global-id, node-id, destination global-id, node-id, tunnel-number. Instead of

 © 2015 IP Infusion Inc. Proprietary

global-ids, both CC and ICC is used. The key used while inserting and searching the tunnel inside the tunnel tree is the combination of source and destination identifiers, which is global-id and node-id.

MPLS TP OAM FSM

All the CFM FSMs are placed inside the CFM library in order to exploit the benefits provided by the library architecture. Once the state change happens because of any events, there is a possibility that, technology specific action that needs to be carried out as part of this state change. Those places are identified and provision to issue the callback will be provided. At the time of initialization the corresponding module, the transport path technology which is interested should register to these callbacks.

OAM Messages

The message generator and receiver functionality for all Y.1731 based MPLS TP OAM is in the line card OAM.

Distributed Layer

The distributed layer APIs and message will be used to send / receive information between line card and control card. A specific data structure and its message is used in the distributed layer. This message will be used to send the configuration data as well as the FSM information from control card to line card.

© 2015 IP Infusion Inc. Proprietary

CHAPTER 8 MPLS-TP Linear Protection Switching

MPLS-TP is a simplified version of MPLS for transport networks with some of the MPLS functions turned off, such as Penultimate Hop Popping (PHP), Label-Switched Paths (LSPs) merge and Equal Cost Multi Path (ECMP). MPLS-TP does not require MPLS control plane capabilities because it enables its management plane to set up LSPs manually. OAM (Operations, Administration, and Maintenance) provides a method for fault management and performance monitoring in both the transport and the service layers to improve their ability to support services with guaranteed and strict Service Level Agreements (SLAs), as well as reducing their operational costs. OAM for MPLS-based transport networks consists of a comprehensive set of fault indication and performance monitoring capabilities.

MPLS-TP LPS Overview

Protection switching uses preassigned capacities between nodes, where the simplest scheme has a single, dedicated protection entity for each working entity, and the most complex scheme may have a number of protection entities shared between numerous working entities. Linear protection provides a rapid and simple protection switching mechanism. In a mesh network, linear protection provides a suitable protection mechanism because it can operate between any pair of points within the network. It can protect against a defect in an intermediate node, a span, a transport-path segment, or an end-to-end transport path. The reversion mode of the protection mechanism may be either revertive or non-revertive. After a service has been recovered. traffic can either be redirected back onto the original working LSP (revertive mode) or it can be left where it is on the recovery LSP (non-revertive mode).

Features

ZebOS-XP MPLS-TP Linear Protection Switching supports the following:

- The ability to create and manage a protection group comprised of a primary and a backup maintenance entity
- Protection Switching Coordination (PSC) state machine
- Management of PSC events
- Primary and backup MPLS-TP Tunnel management
- Management of native services (map-route and VC) over MPLS-TP tunnels that are part of a protection domain
- One-to-one (1:1) and one-plus-one (1+1) protection architectures
- Unidirectional 1:1 and 1+1 protection
- Bidirectional 1:1 protection

Standards Reference

ZebOS-XP supports both IETF and ITU-T standards in the MPLS-TP LPS implementation.

ZebOS-XP supports these IETF standards for LPS:

- RFC 5586 MPLS Generic Associated Channel
- RFC 5960 MPLS Transport Profile Data Plan Architecture
- RFC 6372 MPLS Transport Profile Survivability Framework
- RFC 6378 MPLS Transport Profile Linear Protection

ZebOS-XP supports these ITU-T standards:

- G.8031 Ethernet linear protection switching
- G.8131 Linear protection switching for transport MPLS (T-MPLS) networks

Architecture

The building blocks of the Protection Switching Coordination module are:

- Local Request Logic
- PSC Control Logic
- PSC Message Timer
- Wait to Restore (WTR) timer

The following figure displays the logic flows on a control card and a line card in a distributed architecture.

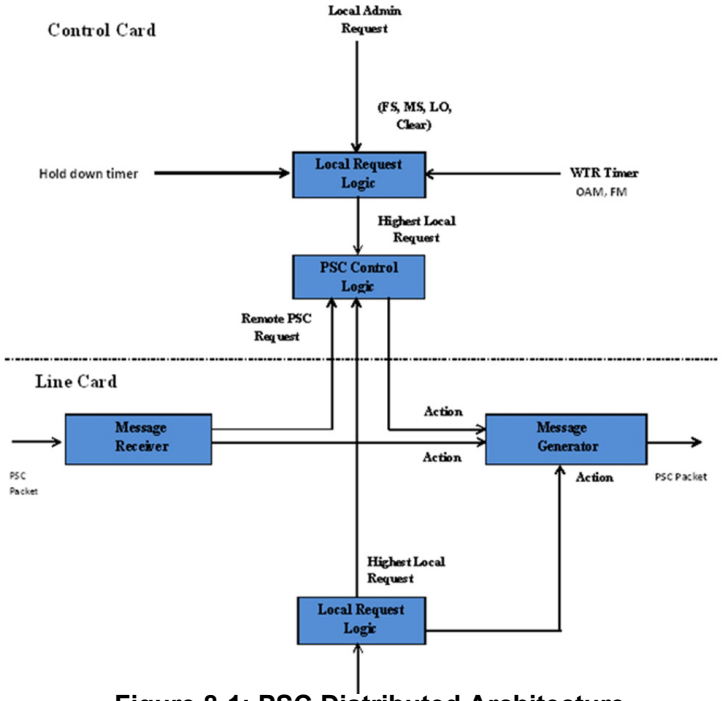

Figure 8-1: PSC Distributed Architecture

Control Card

Control Card (CC) has the major part of the PSC protocol. As part of local request logic, which includes operator commands, the PSC Control Logic and WTR Timer are contained in the control card. The Local Request Logic in the Control Card receives only the operator commands, which include FS, MS, LO or Clear inputs.

Line Card

Line Card (LC) has the PSC message generator, a part of the local request logic that handles input from OAM, and the Remote Request Logic. In order to generate PSC packets, LC needs to have a packet generation interval set. The LC maintains the PSC group data that includes the state of the PSC group, continual-transmit interval, rapid-transmit interval, and others. The timer logic to generate and send the continual and rapid PSC packet runs in the LC OAMD.

© 2015 IP Infusion Inc. Proprietary

Distributed Layer

The distributed layer APIs and messages send and receive information between the line card and the control card.

Event Handling

Basic steps involved in Linear Protection Event handling are outlined in the following sections.

Create and Manage Linear Protection Group

Operator executes the PSC group create commands in the CC. On execution, OAMD does the following:

1. Checks the PSC group is already present in the `aps_group_tree`. If the entry is not there, create a new PSC group and add the same to `aps_group_tree` tree.
2. Sends a group add message to LC using the distributed layer.
3. Sets the state of the PSC group to down, since the association of the primary and backup MEGs are not available.

Additional configuration commands in the protection-group mode are sent as update messages from the CC to the LC so that they are in sync with each other. When an add message comes from the distributed layer, the following occurs:

1. OAMD process running in the LC creates a new `mpls_tp_aps_group_fwd` and adds it to the `aps_group_fwd_name` tree.
2. Checks the operational state of the protection group. If the state is down, then no PSC message is sent.

At this point, if the operational state is UP, invoke the HAL API to create the protection switching group in the hardware. When off-loading to hardware is not supported, start a timer with the value set as continual transmit.

Associate Primary and Backup MEG with Protection Group

Commands are available to configure the primary and backup MEs in a protection group. When you have finished the configuration process, these steps are taken.

1. The following validations are completed:
   - A check in the ME table verifies whether the primary and backup MEs are configured. If the check fails, then an error is returned.
   - Operational state of the ME is validated. If the state is down, then an error is returned, because if the state of the ME is down, then the associated transport path configuration is not complete.
   - MEG type is validated. If the MEG type is not a tunnel MEG then error will be thrown.
   - A check verifies whether this ME is already a part of a protection group. If so, an error is returned.
   - Validate whether the primary and backup tunnel mode are same. If not, an error is returned.
   - In the case of an associated tunnel, all validations are also performed for the forward tunnel.
2. Underlying tunnel source ID is validated against the global ID and the node ID of the node. If they do not match, an error is returned, because protection group configuration is only allowed in an LER.
3. Tunnel destination of both the primary and the backup ME are verified. If they are not, an error is returned.

© 2015 IP Infusion Inc. Proprietary

4. When all checks are successful, the primary and the backup ME pointer in the protection group are updated. The protection group name in the ME data structure is also updated with the group name. In case of an associated tunnel ME, the group name information is updated for both the forward and the reverse ME.
5. Once the association process is complete, the operational status is set to UP, the state of the protection group is set to `MPLSTP_LPS_NORMAL`, and an update message is sent to the LC along with the ME association.
6. OAMD sends an `NSM_MSG_MPLS_TP_APS` message with message type `OAM_NSM_MPLSTP_LPS_CONFIG` to NSM about the primary and the `backup` association with the protection group.
7. To remove primary and backup MEs, an `NSM_MSG_MPLS_TP_APS` message with message type `OAM_NSM_MPLSTP_APS_UNCONFIG` is sent to NSM

Trigger Local PSC Events

Commands are provided in the OAMD to trigger PSC events. Local PSC events that can be triggered by the operator are forced switch (FS), manual switchover (MS), lockout of protection (LO), Clear FS, Clear MS and Clear LO. These triggers can be initiated over the Protection group.

1. If PSC events are triggered in the protection group corresponding to the PG, a name is fetched from the PG tree.
2. PSC FSM is invoked in the control card. The state machine sends the Req Field, Path Field and FPath Field of the PSC packet along with the next state.
3. CC forms the message `aps_group_dist_info` with type as `OAM_MPLSTP_LPS_EVENT` after filling in the information and sends it to the Line Card.
4. CC OAMD sends `NSM_MSG_MPLS_TP_APS` with message type `OAM_NSM_MPLSTP_APS_EVENT` to NSM.
5. LC receives the data from the distributed layer. If the message indicates a change in the protection group, then:
   - HAL API is invoked to inform the HSL (Hardware Service Layer) about the state change.
   - A timer is started with the timeout value equal to the rapid-transmit interval. Upon expiration of this timer, the message generator forms a PSC packet and send it to the sink MEP.

Trigger Remote PSC Events

When the BFD module detects a local signal fail or clear signal fail event, a switchover API with flag value as enable or disable is invoked. This API does the following:

1. If the hold-off timer is configured then the hold-off timer is started.
2. Upon expiration of the hold-off timer, the local event is checked and if the event is SF (signal fail), then the following action is taken:
   - The HAL API is invoked to inform the HSL about the switchover.
   - A timer is started with the timeout value equal to the rapid-transmit interval. Upon expiration of this timer, the message generator forms a PSC packet and send it to the sink MEP. The timer is stopped once it has expired for the third time.
3. The control card OAMD executes the FSM and changes the state appropriately.
   - If the event is a CLEAR SF and the reversion mode of the protection group is revertive then the WTR timer will be started with the configured value. This will be taken care by the PSC FSM. When this timer is getting expired, then it will generate WTR Timer expiry event to the PSC FSM. This timer will be stopped by the events as follows; local LO, local FS an MS, local SF on working and protect, remote LO, remote FS and MS, remote SF on working and protect.

 © 2015 IP Infusion Inc. Proprietary

- The Control Card OAMD sends `NSM_MSG_MPLS_TP_APS` message with message type `OAM_NSM_MPLSTP_APS_EVENT` to NSM.

© 2015 IP Infusion Inc. Proprietary

CHAPTER 9 MPLS-TP Ring Protection Switching

MPLS-TP Ring Protection Switching (MRPS) is built over ring topologies similar to the topologies in Synchronous Optical Networking (SONET) and Synchronous Digital Hierarchy (SDH) networks.The MRPS topology provides an effective and fast recovery time, efficient protection mechanism, and high bandwidth utilization by using the packet switching statistical multiplexing.

System Features

MRPS is equivalent to SDH Manual Switch (MS) shared protection ring architecture. The protection ring consists of two counter-rotating rings, transmitting in opposite directions relative to each other. Both rings carry working and protection traffic.

ZebOS-XP MPLS-TP RPS supports the wrapping scheme which defines ring protection switching for link and node failures over point-to-point (p-t-p) and point-to-multipoint(p-t-mp) Label Switched Paths (LSP).

And also MRPS supports the following characteristics.

- Switching types
- Operation types

The Wrapping Ring Protection Scheme

The Wrapping technique defines the mechanism to send out an Automatic Protection Switching (APS) request from the node that is detecting a failure to the node adjacent (opposite to the failure) to the failure. The APS request is transmitted over the APS communication protocol.

When a node detects a failure or receives an APS request through APS protocol addressed to this node, the traffic of all working LSPs/tunnels transmitted towards the failed span is switched to the protection LSPs/tunnels in the opposite direction (away from the failure).

This traffic travels around the ring to the other node (adjacent to the failure) where it is switched back onto the working LSPs/tunnels. The nodes that performed the protection switching revert to the normal traffic flow when the failure or APS request is cleared.

For each normal or working MPLS-TP LSP/tunnel, the protection LSP/tunnel MUST be established in the opposite direction though all nodes in the ring. Labels assigned for the protection LSPs/tunnels must be associated with the labels assigned for working LSPs/tunnels to allow proper traffic switching between the working and protection LSPs/tunnels.

p-t-p LSP Failure example

The following is an example of a p-t-p LSP wrapping protection switching scheme with the label allocation.

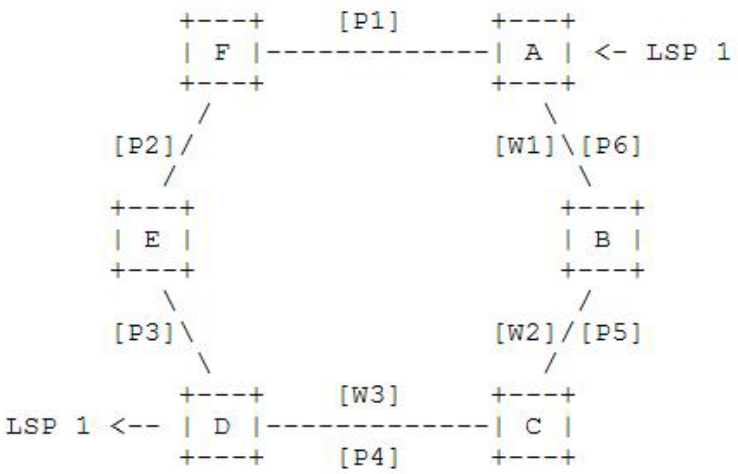

Figure 9-1: p-t-p LSP failure

Working labels	A[W1]->B[W2]->C[W3]->D
Protection labels	A[P1]->F[P2]->E[P3]->D[P4]->C[P5]->B[P6]->A
Working and protection labels association	W1]<->[P6] [W2]<->[P5] [W3]<->[P4]

p-t-p LSP link failure example

The following is an example of a p-t-p LSP link failure.:

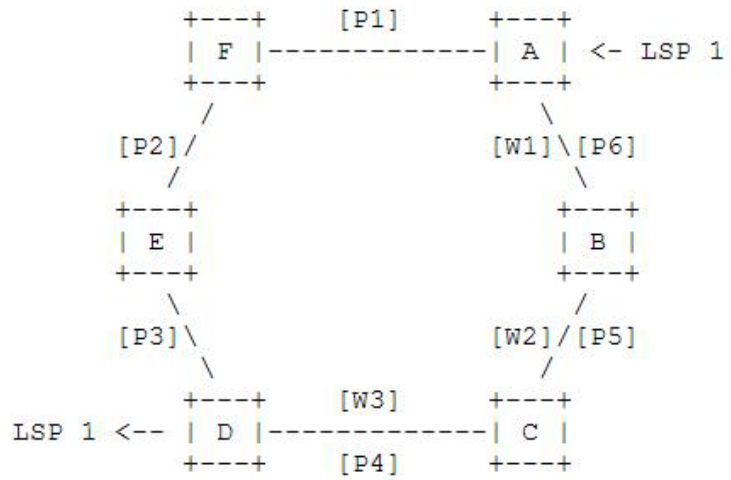

Figure 9-2: p-t-p LSP link failure

When the failure occurs between the nodes B and C, these nodes send APS request to each other around the ring. Node B switches the traffic of LSP 1 from working label [W1] to the protection label [P6] in the opposite direction

© 2015 IP Infusion Inc. Proprietary

counterclockwise. This traffic travels around the ring to the node C where it is switched from protection label [P4] to the working label [W3] and sent to the node D where it is dropped from the ring.

Traffic flow and labels used when the link failure occurs	A[W1]->B[P6]->A[P1]->F[P2]->E[P3]->D[P4]->C[W3]->D

p-t-p LSP node failure example

The following is an example of p-t-p LSP node failure.

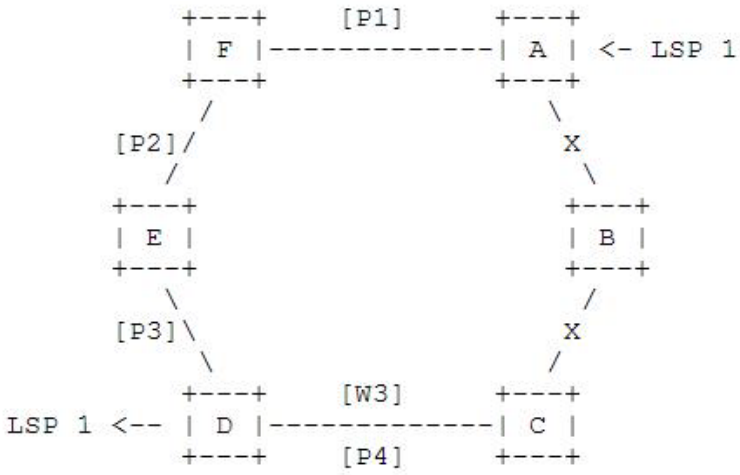

Figure 9-3: p-t-p LSP node failure

When node B fails or becomes isolated because of two failed links, nodes A and C send APS request to each other around the ring. Node A switches the traffic of LSP 1 to the protection label [P1] in the direction opposite to normal flow. This traffic travels around the ring to the node C where it is switched from the protection label [P4] to the working label [W3] and sent to the node D where it is dropped from the ring.

Traffic flow and labels used when the node B failure occurs	A[P1]->F[P2]->E[P3]->D[P4]->C[W3]->D

Switching types

MRPS mechanism supports bi-directional protection switching type. In bi-directional switching, the traffic passing in both directions of the monitored MPLS-TP section layer is switched to protection LSPs/tunnels.

Operation types

MRPS mechanism supports revert protection operation type, where the traffic will return to (or remains on) the working LSPs/tunnels after the failure or APS request is cleared.

Note: The draft recommends the usage of APS and ring map configurations at every node within the protection ring. However, the APS payload structure and the method of configuring the ring maps on the ring nodes are yet to be defined by the draft.

- Hence, this feature implementation will not provide support for the APS protocol.In order to support the “Steering” ring protection scheme, usage of APS protocol is a must. Since, APS will not be supported in this feature release, support for “Steering” ring protection scheme will not be provided as well.
- However, there is still a way to support the "Wrapping" solution without APS provided, the support for manual switching commands like lockout, manual-switch and forced-switch are left out. This is the approach that will be adopted in this feature implementation.

- This approach uses the data-link section-layer OAM, CC, to detect signal failures in the link between any two ring nodes. A failure in the link between two ring nodes would result in CC faults on nodes at either ends of the link. The node identifying the CC fault switches the traffic to the protection path. This traffic travels around the ring to the other node (adjacent to the failure) where it is switched back onto the working path.

© 2015 IP Infusion Inc. Proprietary

CHAPTER 10 MPLS Non-Stop Forwarding

This chapter describes the MPLS-NSF (non-stop forwarding) feature for ZebOS-XP.

MPLS-NSF Overview

NSM hosts an MPLS RIB table, such as incoming label map (ILM), next hop label forwarding entry (NHLFE), FEC-to-NHLFE (FTN) or cross-connect (XC). It passes information to the MPLS forwarder, which then places it in the forwarding information base (FIB) table.

The MPLS control-plane requires two instances, which includes an active and a standby instance. Typically, a user configures an active instance on the active control processor, and then configures a standby instance on a standby control processor. The active instance establishes sessions, exchanges routing information with peers, and builds RIB/FIB tables. The standby instance maintains a state for stateful components and becomes active following a control-plane switchover. Moreover, the standby instance does not exchange routing information with external peers nor does it process data packets. In effect, the standby instance acquires sufficient data (state) from the active instance, so that after a control-plane switchover it becomes active to recover the control-plane state and validate the forwarding state.

When there is a fault in the active instance, the MPLS control-plane needs to be certain that the data path is not broken, so that the MPLS forwarder can keep forwarding MPLS packets based on the IPs stored FIB table.

Checkpoint Abstraction Layer

MPLS-NSF utilizes the checkpoint abstraction layer (CAL) in ZebOS-XP to “checkpoint” data from both the active and standby instances. High-availability (HA) for MPLS supports two different models: simplex-active and active-standby.

Simplex Active Mode

In this mode, redundancy is built within a single node. If (for any reason) a process daemon is about to fail, CAL ensures that it restarts and comes back up with the data that was checkpointed earlier.

Active-Standby Mode

In this mode, there is a 1:1 redundancy where the active instance checkpoints data, which is then asynchronously synced onto the standby machine. This mode sustains even if the active and standby processors are on the same chassis and sharing the same line card or if they are in different chassis with different line cards. CAL checkpoints all data for the MPLS forwarder in the respective line cards. In this implementation, the ADB (application database) is the MPLS data structure within NSM and in the MPLS FIB database. Part of this information form the CDB (checkpoint database), which will be synced into the standby.

MPLS Tables

The following subsection describes the supported tabled used by MPLS-NSF.

FTN Table

The FTN table maps each FEC to a set of NHLFE objects. This table is used when forwarding packets that arrive unlabeled, but need to be labeled before being forwarded.

ILM Table

The ILM table maps incoming label to a set of NHLFE objects. In ZebOS-XP, the RIB entry structure is located in the ilm_entry file.

NHLFE table

The NHLFE table is used when forwarding a labeled packet. This table contains the following information:

- Nexthop of a packet
- Operation required to perform on the label stack for a packet, including one of the following operations:
  - Replace the label at the top of the label stack with a specified new label (swap)
  - Pop the label stack
  - Replace the label at the top of the label stack with a specified new label. Then push one or more specified new label onto the label stack

Cross-connect (XC) Table

The cross-connect (XC) table connects an incoming label to an outgoing label. In ZebOS-XP, the XC table is referenced and linked from all other tables (as previously described). An FTN entry has a pointer reference to the XC table. In addition, ILM entries have a cross-connect table index. The XC table hosts the NHLFE table to reference all NHLFE entries.

LDP Graceful Restart

MPLS NSF uses the LDP Graceful Restart process to execute control plane switchover. In the Active instance, LDP stops gracefully. Then switchover begins and LDP starts in the standby instance of the graceful restart mode. The standby instance then has all MPLS RIB synced from the active instance and it is populated back into LDP.

When an LDP graceful restart is triggered, LDP sends a message with reconnect and recovery timer values to NSM to preserve the RIB and FIB entries. When LDP disconnects from the NSM server, NSM checks whether a restart option was set for a client. If a restart option was set, then any client-specific entry in the MPLS RIB and label pools are marked as stale. In the peer router, the entries are also marked as stale for a period called the “negotiated reconnect timeout”. When LDP restarts, LDP registers any NSM client with the NSM server. Once NSM receives this request from LDP, it passes the staled label pools and ILM-related information back to LDP and stops the preserve timer.

LDP proceeds with re-establishing LDP. When allocating a label for an FEC, LDP does a lookup in the stale database to determine if any stale entry exist. If an entry is found for a corresponding FEC, the same label is used. If no preserved label is found for an FEC, LDP allocates a new label and FEC-label bindings are exchanged with peer routers. The forwarding table is updated accordingly. The LDP session is expected to restart within the recovery time. If it fails to do so, then the FIB and RIB entries are deleted. Recovery time is applicable only after restart. Recovery time is the maximum time until which stale entries are maintained after the session re-initiates between the restarting router and its peer. Once an LDP session is reestablished, LDP sends FTM and ILM entries to NSM. If a stale entry is present, then NSM unsets the stale flag.For details about the Label Distribution Protocol graceful restart feature, see the *Label Distribution Protocol Developer Guide*.

RSVP-TE Graceful Restart

The RSVP-TE graceful restart mechanism helps minimize the traffic downtime after a restart of the control plane. The forwarding state is maintained and traffic is uninterrupted during this period. After restart, the node refreshes its RSVP session states with the help of any neighboring LSR, so that the forwarding state is valid and no new resource reservations are necessary.

© 2015 IP Infusion Inc. Proprietary

Initial Capability Exchange

In the first phase of RSVP-TE graceful restart, any neighboring LSR advertises its graceful restart capability to each other using a RESTART_CAP object in Hello messages. The RESTART_CAP object includes two parameters: restart time and recovery time. The restart time specifies the duration (in milliseconds) in which the sender is expected to reestablish a Hello communication with the receiver after restarting its RSVP-TE control plane. If a sender's control plane restart expects to take an indeterminate time and its forwarding plane remains unaffected across a restart, then the sender advertises a value of "0xffffffff" as the restart time, which indicates an indefinite restart period.

RSVP-TE Control Plane Restart

The RSVP-TE control plane restart is the second phase of RSVP-TE graceful restart. During this phase, the restarting LSR continues to forward traffic (across the control plane restart) using the preserved MPLS forwarding state. When an LSR detects that a neighbor restarted and the neighbor has previously indicated its capability to preserve the MPLS forwarding state using the RESTART_CAP object, then the LSR retains its RSVPTE control plane and MPLS forwarding state for all established LSPs that traverse links between the LSR and the restarting neighbor. In addition, the LSR waits until the RSVP-TE reestablishes its Hello communication with its neighbors.

Reestablishment of Hello Communication

The reestablishment of hello communication is the third phase of RSVP-TE graceful restart. It begins when the restarting LSR, after restarting its control plane, is ready to reestablish Hello communication with its neighbors. After restarting, the LSR checks whether it was able to preserve its MPLS forwarding state across the control-plane restart. If it cannot, the LSR sends the recovery time as 0 (zero) in the Hello message RESTART_CAP object. If the LSR was able to preserve its state, then it sends a non-zero value as the recovery time. Once neighbors receive the hello message, they enter the recovery phase and update the RSVP state with the neighbor.

System Architecture

The protocol daemons receive a label pool block from the label pool managed by NSM. Each protocol establishes a session with its peer and negotiates for incoming and outgoing labels, FEC mapping, LSPs and Tunnel establishment. In addition, each protocol communicates all of its information to NSM, which stores it in the MPLS RIB tables of FTN, ILM, XC and NHLFE. NSM then "pushes" these tables into the FIB via the MPLS forwarder module. Once done, the traffic is switched across using the FIB tables.

NSM MPLS Tables

The NSM MPLS RIB consists of four tables: NHLFE table and Cross-connect (XC) Table. The XC table links to all remaining tables. Each table is formed and populated by various triggers, as depicted in Figure 10-1.

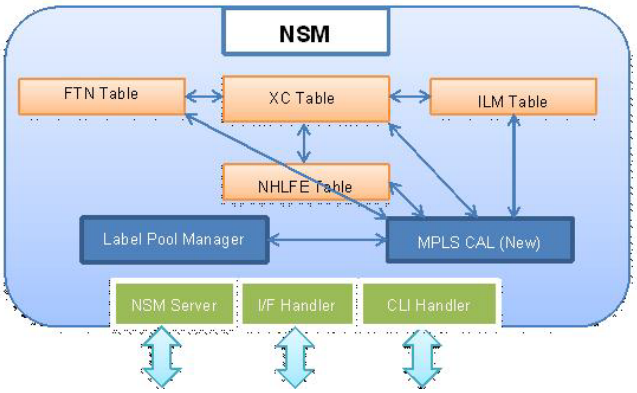

Figure 10-1: NSM MPLS Tables

NSM Server

The NSM server module receives all IPCs from the protocol daemons, such as LDP or RSVP, and modifies the RIB table accordingly.

Interface Handler

The interface handler module manages the processing involved when there is a state change for an interface; that is, when it goes down and comes back up. When an interface goes down, all protocol-initiated RIB entries are deleted. However, all statically configured entries are retained, but are kept stale. When the interface comes back up, stale entries are reactivated.

CLI Handler

The CLI handler module processes all CLI-initiated transactions. Using CLI commands, all FTN and ILM entries can be explicitly added and any LSP can be statically configured. Note that statically-configured LSPs may take non-optimal paths.

Label Pool Manager

The Label Pool Manager module manages the label space. In ZebOS-XP, label space is a per-interface type. Labels are grouped in small blocks of bucket size 640. When any protocol or CLI command requests labels, the label pool managed allocates the next free block from the pool.

MPLS CAL

This module processes all checkpointing for the MPLS-RIB, as well as other essential information, including abel pools and protocol graceful-restart-related information. It extracts information from databases and communicates the information to its peer in standby mode. In the standby interface, the information is stored in its databases. In addition, MPLS CAL module processes all actions required for synchronization.

© 2015 IP Infusion Inc. Proprietary

CHAPTER 11 Layer 2 Virtual Circuit

This chapter describes the Layer 2 Virtual Circuit (VC) feature for ZebOS-XP.

Layer 2 VC Overview

The L2 VC module is a part of the Label Distribution Protocol (LDP) module. The NSM provides support for managing VCs. The L2 VC module sets up virtual circuits for transporting Layer 2 protocols across an MPLS network. ZebOS-XP supports Ethernet, Ethernet VLAN and PPP Layer 2 protocols. The MPLS Forwarder supports encapsulation of Ethernet frames over MPLS Virtual Circuits.

Terminology

Virtual Circuits

Virtual Circuits are MPLS LSP tunnels used to carry Layer 2 protocol data units (PDU) over MPLS networks. The ingress and egress nodes of the tunnel use an extended LDP protocol to exchange VC labels to create a virtual circuit. These labels are only visible at the end points of the circuit. A VC is identified by its Virtual Circuit ID (VC-ID). It is bi-directional in nature with a separate VC label allocated for each direction of traffic flow. This makes it possible to extend a LAN over a shared WAN infrastructure, enabling service providers to provide a transparent LAN service (TLS) over the shared public network infrastructure.

Virtual Circuit ID

Virtual Circuit Identifier (VC-ID) is a 32-bit number that uniquely identifies a Virtual Circuit. To create a virtual circuit, the same VC-ID needs to be configured on both PE routers forming the end-points of a Virtual Circuit.

Virtual Circuit Labels

VC labels are exchanged between the two end nodes of a VC, one label for each direction of traffic flow. These labels are not visible to intermediate MPLS nodes.

Tunnel Labels

Tunnel labels are required to transport a packet across an MPLS network from the ingress node to the egress node of a VC. A tunnel LSP may be shared by more than one VC. Tunnel labels are exchanged between adjacent MPLS nodes using a label distribution protocol; in the case of ZebOS-XP, LDP is used to create tunnel LSPs.

Layer 2 PDU

The Layer 2 PDU is encapsulated using MPLS labels. ZebOS-XP MPLS VC supports LDP extensions for Ethernet, PPP and Ethernet VLAN. The MPLS Forwarder supports encapsulation of Ethernet PDUs only.

Control Word

Control word refers to the extra information that needs be carried for transportation of some encapsulated Layer 2 PDUs over the backbone. The control word is optional for Ethernet, PPP and Ethernet VLAN.

Traffic Flow Across Virtual Circuits

Figure 11-1 on page 70 depicts packet flow across an MPLS virtual circuit. The dotted line defines the VC established between two PE devices. The solid lines depict actual data paths between two CE devices. The diagram shows the encapsulation of Layer 2 PDUs as they are carried over a virtual circuit.

When the two PE routers are configured to create an MPLS VC, they use the extended LDP protocol to exchange VC labels. LDP is also used to create a tunnel LSP between the two PE routers. The layer 2 PDU coming from CE is encapsulated by pushing the VC label. If the two PE routers are separated by an MPLS cloud, then another label (tunnel LSP label) is pushed on top of VC label. The labeled packet is then forwarded over the outgoing interface. The packet traverses the shared MPLS infrastructure, where only the top label (tunnel label) is used by intermediate nodes for forwarding. The egress PE router for the VC pops the VC label and the de-encapsulated Layer 2 PDU is forwarded to the CE network. Since the VCs are bi-directional, the same procedure is repeated to forwarded packets coming from the other end of circuit.

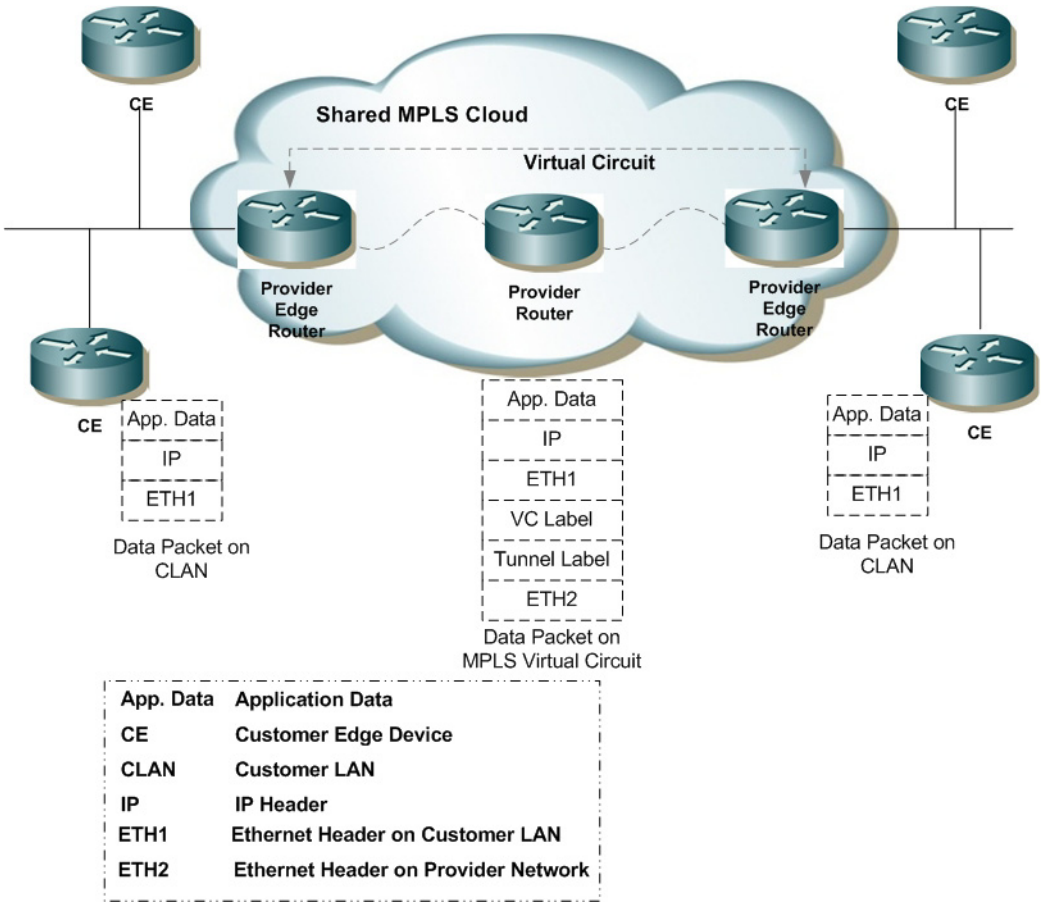

Figure 11-1: Virtual Circuit Traffic Flow

MPLS Virtual Circuit Routing and Forwarding Process

MPLS Virtual Circuit routing and forwarding involves the following steps:

1. Service providers supply MPLS based virtual circuits from PE routers that are directly connected to a customer network.

© 2015 IP Infusion Inc. Proprietary

2. Each PE router maintains two VC labels for each virtual circuit, one for each direction of traffic flow. The VC label used to forward traffic to the other end of the circuit is associated with the interface bound to the virtual circuit. The label used to forward traffic to the local customer network is stored in the ILM table associated with the incoming interface.

3. After the interface is bound to a VC, PE routers exchange VC labels for bidirectional traffic flow using extended LDP. Both PE routers must be configured with the same virtual circuit ID (VC-ID). The ID is used to uniquely identify virtual circuits.

   Note: An interface can be bound to only one virtual circuit, and a virtual circuit cannot be bound to more than one interface.

4. Ingress PE router encapsulates the L2 PDU received from customer with two labels. The top label is the LSP label received from the IP next-hop table for the remote PE router's loopback address. The inner label is the VC label received from the remote PE router for the corresponding VC.

5. The provider MPLS network switches the VC packets based on the top LSP label only. This label is used as a key to lookup in the ILM table bound to the interface on which packets are received. If there is an outbound label, the incoming label is swapped and the packet is forwarded to the next hop. If the MPLS router is the penultimate router for the VC, it pops the LSP label and forwards the packet with only VC label to the VC egress PE router.

6. The egress PE router pops the VC label and transmits the L2 PDU on the outbound interface determined from the ILM entry corresponding to the VC label.

NSM Extensions

NSM supports the configuration of virtual circuits with operator commands you enter at the ZebOS-XP CLI. The two principal commands are used to add, update or delete virtual circuits.

Note: For details about the commands used to add, update or remove virtual circuits, see the *Multi-Protocol Label Switching Command Reference*.

Messages between NSM and LDP convey information about VCs to the LDP module. A client handler manages the messages and calls appropriate callbacks registered by the client protocol.

- VC add: A VC add message is sent to the client by NSM whenever a VC is added or an existing VC is updated. A VC is always bound to a specific interface and an add message is only sent when the interface status is UP.
- VC delete: A VC delete message is sent to the client under any of the following circumstances:
  - A configured VC is deleted
  - The interface to which a given VC is bound goes down
  - The interface to which a given VC is bound is deleted

  Note: An interface cannot be bound to both VRF and VC at the same time.

LDP Extensions

In LDP, you need to configure a targeted peer to bring up a targeted session for the VC peer. The following extensions in ZebOS-XP LDP support MPLS virtual circuits:

- Encoding and decoding for Virtual Circuit FEC element as defined by MPLS-VC
- Processing of VC add/delete messages from NSM
- A VC table in LDP stores information about all configured Virtual Circuits. A 32-bit number (the VC-ID) is the key to this table.

- An LDP session is created with the remote peer of the virtual circuit in `Downstream Unsolicited` mode with `Liberal Label Retention` option. This session is used by virtual circuits to exchange messages with a remote Virtual Circuit peer.
- The VC module manages establishment of bi-directional LSP and contains all associated data structures and processing logic.
- The same session may be used by more than one Virtual Circuit to exchange label information. A session maintains a list of Virtual Circuits using that session.

© 2015 IP Infusion Inc. Proprietary

CHAPTER 12 DiffServ-TE

This chapter describes the DiffServ (Differentiated Services) Traffic Engineering (DiffServ-TE) feature.

DiffServ-TE Overview

DiffServ (Differentiated Services) and MPLS Traffic Engineering (TE) provide different benefits to service provider's networks but have their own limitations. DiffServ is used in networks to provide multiple scalable classes of services to clients. However, it does not have the ability to use network transmission resources efficiently. MPLS Traffic Engineering is used in networks to provide optimization of transmission resources (bandwidth reservation and constraint-based routing). However, it can only provide this capability at one aggregate level; there is no classification of the network bandwidth.

To achieve fine-grained optimization of transmission resources and further enhanced network performance and efficiency, it is desirable to perform traffic engineering at a per-diffserv-class level instead of at an aggregate level. DiffServ-aware Traffic Engineering (DiffServ-TE) maps the traffic from a given DiffServ Class of Service (CoS) to a separate LSP, and allows traffic to utilize resources available to the identified class on both shortest paths and non-shortest paths. It follows paths that meet engineering constraints that are specific to the class.

Terminology

Class-Type

Class-Type (CT) is a set of Traffic Trunks crossing a link that is governed by a specific set of bandwidth constraints. CT is used for link bandwidth allocation, constraint-based routing and admission control. A given Traffic Trunk belongs to the same CT on all links.

TE-Class

TE-Class is a combination of a Class-Type and a preemption priority allowed for that Class-Type. This means that an LSP transporting a Traffic Trunk from a specified Class-Type can use the preemption priority as a set-up priority or as a holding priority or as both. This mapping is referred to as:

```
TE-Class[i]  <-->  < CTc , preemption p >
```

where `0 <= i <= 7, 0 <= c <= 7, 0 <= p <= 7`

Bandwidth Constraint

Existing TE mechanisms only allow constrain-based routing of traffic based on a single Bandwidth Constraint (BC) common to all classes of service, which does not satisfy the needs of DiffServ-TE. In a DiffServ-TE implementation, different bandwidth constraints are defined for different CoS.

Bandwidth Constraint Model

The Bandwidth Constraint Model is a set of rules defining:

- the maximum number of Bandwidth Constraints; and
- the Class-Types that each Bandwidth Constraint applies to and how.

  For example, an operator can define a model with one separate Bandwidth Constraint per Class-Type. This model is referred to as the `Maximum Allocation Model (MAM)` and is defined by:

  - MaxBC= MaxCT
  - Each value of b in the range `0 <= b <= (MaxCT - 1)`: Reserved BC (CTb) <= BCb

ZebOS-XP DiffServ-TE Implementation

The ZebOS-XP DiffServ-TE implementation involves NSM, RSVP-TE, CSPF, and OSPF.

NSM

ZebOS-XP DiffServ-TE supports eight Class-Types, eight Bandwidth Constraints and eight TE-Classes. Using the NSM, operators can configure a CT name for each CT.

Operators can also configure corresponding CT mapping and preemption for each TE-Class. The configured TE-Classes and its related Class-Types are passed to the RSVP-TE protocol and the configuration is used for setting up DiffServ-TE LSPs. This information is passed by a new IPC message: `NSM_MSG_DSTE_TE_CLASS_UPDATE`.

The ZebOS-XP DiffServ-TE implementation supports Maximum Allocation Model (MAM) through its QoS agent in the NSM. The MAM implementation is fully compliant with the IETF draft: `draft-ietf-tewg-diff-te-mam-00.txt`. It also allows the operator to configure other bandwidth constraint models (such as, the Russian Doll model - `draft-ietf-tewg-diff-te-russian-03.txt`) per interface through NSM while customer needs to develop the other bandwidth constraint model implementation through their QoS module by themselves. The configured Bandwidth Constraint mode is passed to OSPF by an updated IPC message: `NSM_MSG_INTERFACE_UPDATE`.

When a DiffServ-TE LSP is setting up and the corresponding bandwidth is changed, NSM passes the information to CSPF by an updated IPC message: `NSM_MSG_INTERFACE_UPDATE`.

Updated IPC Message	NSM uses this message to
`NSM_MSG_DSTE_TE_CLASS_UPDATE`	Pass information about configured TE-Classes and its related Class-Types to RSVP-TE.
`NSM_MSG_INTERFACE_UPDATE`	Pass the configured Bandwidth Constraint mode to OSPF.
`NSM_MSG_INTERFACE_UPDATE`	Pass information about changed bandwidth during an DiffServ-TE LSP setup. to CSPF.

© 2015 IP Infusion Inc. Proprietary

RSVP-TE

To set up DiffServ-TE LSPs an additional object, `CLASSTYPE`, is added to RSVP Path messages. The `CLASSTYPE` object process is IETF compliant. When Constrained Shortest Path First (CSPF) is enabled, RSVP-TE sends an updated IPC message `ROUTE_REQUEST` to request CSPF to compute Constraint Based routing for the LSP. In case of DiffServ-TE, RSVP-TE passes related DiffServ-TE information (Bandwidth Constraint, `CLASSTYPE`, etc.) to CSPF. When RSVP-TE receives the IPC message `NSM_MSG_DSTE_TE_CLASS_UPDATE` from NSM, it processes this message and performs appropriate actions (bring up/down) to adjust the status of the DiffServ-TE LSPs.

OSPF

To pass bandwidth constraint information from a local router to other routers, an additional sub-TLV, Bandwidth Constraint sub TLV, is defined in the TLV line. The existing `Unreserved Bandwidth` sub-TLV is assigned a new semantic. The 8 bandwidth values of this sub TLV correspond to the unreserved bandwidth for each of the TE-Class in DiffServ-TE instead of corresponding to each preemption priority as per existing TE. The process of these two sub-TLVs are fully complaint with the IETF draft: `draft-ietf-tewg-diff-te-proto-04.txt`. When OSPF receives an IPC message `NSM_MSG_INTERFACE_UPDATE` from NSM, OSPF updates its Bandwidth Constraint mode information and bandwidth constraint information according to the information specified in this message.

CSPF Extensions

In CSPF, the constraint routing algorithm is updated to enable the calculation of a constraint based route according to the DiffServ-TE information received from signalling protocols.

DiffServ-TE Application

DiffServ-TE is primarily used to provide a finer granularity in bandwidth constraint definitions in DiffServ-aware networks. Whereas, traditional bandwidth models do not provide control over bandwidth on a per traffic class basis. For example, in a non DiffServ-TE network where voice and data traffic are supported, if a 100 MB of bandwidth is available through the 8 traditional priority levels, it is the responsibility of the operator to ensure that a data trunk is not able to consume more than a specified bandwidth (say 20 MB), so that a certain bandwidth (say 80 MB) is always available for voice trunks. However, in case of a mis-configured data trunk that has a higher priority than a voice trunk, it will preempt a voice trunk when enough bandwidth is not available. In this case, the operator does not have any control over how the bandwidth is distributed among different types of trunks and the only governing factor is priority of the trunk.

In a DiffServ-TE network, the operator can assign a chunk of bandwidth each to voice trunks and data trunks in such a way that data trunks always consume bandwidth out of the bandwidth pool set aside for data traffic. Here the operator defines bandwidth constraints on a per class basis making the governing factor a combination of priority and Class-Types. Additionally, a voice trunk can preempt a less important voice trunk without disturbing data trunks while consuming bandwidth only out of the bandwidth pool.

In a DiffServ-TE network, the operator must define bandwidth constraints based on the Class-Type. Each Class-Type combined with a priority level makes up one of 8 supported TE classes on a DiffServ-TE aware router. The operator then configures trunks based on combinations of Class-Type and priority that have been defined as valid and supported TE classes.

© 2015 IP Infusion Inc. Proprietary

CHAPTER 13 Structure Agnostic TDM over Packet

This chapter describes the Structure Agnostic TDM over Packet (SAToP) feature for ZebOS-XP.

SAToP Overview

Telecoms are based on time division multiplexing (TDM) technologies (T1/E1 and SONET/SDH). This technology is best suited to transmit constant bit rate traffic, such as digitized voice and video packets. TDM technologies support small transmission delay and small transmission delay variation, which are two key parameters for voice quality and video transmission. As mobile back-haul transitions to packet-type infrastructures, it is important to efficiently carry TDM signals on a packet switched network. This requires the ability to convert TDM to packet formats while maintaining accurate synchronization information for these circuits. In addition, a TDM requires low latency and jitter for reliable transmission, which can only be achieved by guaranteeing adequate bandwidth, prioritization and buffering along the end-to-end path to minimize packet losses, delays or reordering.

Structure Agnostic TDM over Packet (SAToP) is an encapsulation method that is used to transparently carrying a TDM signal without processing the DS0 time-slot. SAToP creates packets for an incoming TDM stream, which includes a suitable header. In structure-agnostic transport, any structural overhead present is transparently transported along with the payload data. In addition, the encapsulation provides no mechanisms for its location or utilization.

Emulated Services

The specification describes edge-to-edge emulation of the following TDM services described in [G.702]:

- E1 (2048 kbit/s)
- T1 (1544 kbit/s) - also known as DS1
- E3 (34368 kbit/s)
- T3 (44736 kbit/s) - also known as DS3

Protocols used for emulation of services do not depend on the method in which attachment circuits are sent to PEs.

SAToP Packet Format

The path MTU is the limiting factor for the total size of a SAToP packet sent between two PEs. The SAToP header consists of a SAToP control word that is mandatory and may contain a RTP header. If present, the RTP header will immediately follow the SAToP control word in all cases, except for UDP multiplexing.

Using the SAToP control word allows users to do the following:

- Detect of packet loss or mis-ordering.
- Differentiate between a PSN (packet-switching network) and an attachment circuit problem, which might have been caused for the outage of an emulated service.
- Conserve PSN bandwidth by not transferring invalid data (AIS).
- Signal fault detected at the PW egress to the PW ingress.

Supported modules

This section describes the ZebOS-XP modules that support SAToP implementation.

NSM

Network service Manager (NSM) manages the information related to routes. NSM shares this information to other modules of ZebOS-XP using asynchronous sockets created at startup. It handles the creation and maintenance of pseudowires. NSM will create virtual interfaces and attach the virtual interfaces to the virtual circuit created. It will also process the status information received from peers and its own.

LDP

Label Distribution Protocol (LDP) is a routing component of MPLS technology that distributes the labels in MPLS environment. LDP can be used to distribute the inner label (VC/VPN/service label) and outer label (path label) in MPLS. For inner label distribution, targeted LDP (tLDP) is used. The LDP protocol runs as a separate daemon in ZebOS-XP, and uses NSM services to obtain routing information.

HAL

The Hardware Abstraction Layer (HAL) module is library used by daemons to communicate to L2SWFW, hal and HSL modules. NSM needs to interact with HAL for error notifications and statistics. Though this is required, it will not be implemented as part of this development.

SaTOP APIs

The following subsection describes the supported SaTOP API functions.

nsm_server_send_mpls_l2_circuit_add

Use this function to send the values of payload_bytes, bit_rate, tdm_options and other virtual circuit parameters. NSM sends these parameters to LDP using this function to help set up the pseudowire feature.

Syntax

```
s_int32_t
nsm_server_send_mpls_l2_circuit_add (struct nsm_master *nm,
                                     struct nsm_server_entry *nse,
                                     struct nsm_mpls_circuit *vc,
                                     u_int16_t vc_type)
```

Input Parameters

`*nm`	A pointer to NSM master structure
`*nse`	Set the NSE pointer and size.
`*vc`	VC name
`vc_type`	Port used for a VC type

Output Parameters

None

© 2015 IP Infusion Inc. Proprietary

Return Values

0

nsm_if_mpls_l2_circuit_bind

Use this function to add or delete a new NSM interface type and create a mapping between PWs and TDM interfaces.

Syntax

```
int
nsm_if_mpls_l2_circuit_bind (struct cli *cli, char *vc_name, char *vc_type,
                             u_int16_t vlan_id, u_char vc_mode)
```

Input Parameters

`*cli`	Specifies the instance of the CLI structure.
`*vc_name`	Port used for a VC name
`*vc_type`	Port used for a VC type
`vlan_id`	VLAN ID
`vc_mode`	Port used for a VC mode

Output Parameters

None

Return Values

NSM_SUCCESS

NSM_FAILURE

nsm_mpls_l2_circuit_bind_by_ifp

This function is to be used by the interface code to bind to a layer-2 circuit.

Syntax

```
int
nsm_mpls_l2_circuit_bind_by_ifp (struct interface *ifp, char *vc_name,
                                 u_int16_t vc_type, u_int16_t vlan_id,
                                 u_int16_t vc_mode)
```

Input Parameters

`*ifp`	Pointer to the interface
`*vc_name`	Port used for a VC name
`vc_type`	Port used for a VC type
`vlan_id`	VLAN ID
`vc_mode`	Port used for a VC mode

Output Parameters

None

Return Values

NSM_ERR_VPN_EXISTS_ON_IF

NSM_ERR_NOT_FOUND

NSM_ERR_VC_UNSUPPORTED_MODE_FOR_TP

NSM_TDM_ERR_PAYLOAD_NOT_CONFIGURED

nsm_tdm_payload_bytes

Use this function to set the TDM parameter for payload bytes.

Syntax

```
int
nsm_tdm_payload_bytes(struct interface *ifp ,u_int16_t ifpayload)
```

Input Parameters

`*ifp`	Pointer to the interface
`*ifpayload`	

Output Parameters

None

Return Values

NSM_SUCCESS

NSM_FAILURE

nsm_tdm_set_error_period

Use this function to set the error-set period.

Syntax

```
int
nsm_tdm_set_error_period (struct interface *ifp, u_int16_t error_time, u_int8_t type2 )
```

Input Parameters

`*ifp`	Pointer to the interface
`*error_time`	
`type2`	

Output Parameters

None

Return Values

NSM_SUCCESS

© 2015 IP Infusion Inc. Proprietary

nsm_tdm_set_jitter_buffer_size

Use this function to configure the jitter-buffer size.

Syntax

```
int
nsm_tdm_set_jitter_buffer_size(struct interface *ifp ,u_int8_t buf_size)
```

Input Parameters

`*ifp`	Pointer to the interface
`*buf_size`	Buffer size

Output Parameters

None

Return Values

NSM_ERR_NOT_FOUND

NSM_SUCCESS

nsm_tdm_virtual_interface_set

Use this function to set the TDM virtual interface.

Syntax

```
int
nsm_tdm_virtual_interface_set (u_int32_t vr_id , u_int32_t tid)
```

Input Parameters

`vr_id`	Virtual router ID
`tid`	Tunnel ID.

Output Parameters

None

Return Values

NSM_SUCCESS

NSM_API_SET_ERR_MASTER_NOT_EXIST

nsm_tdm_virtual_interface_unset

Use this function to unset the TDM virtual interface.

Syntax

```
int
nsm_tdm_virtual_interface_unset (u_int32_t vr_id , int tid)
```

Input Parameters

`vr_id`	Virtual router ID
`tid`	Tunnel ID.

Output Parameters

None

Return Values

NSM_SUCCESS

nsm_tdm_rtp_hdr

Use this function to set the real-time header.

Syntax

```
int
nsm_tdm_rtp_hdr(struct interface *ifp , bool_t rtp_hdr)
```

Input Parameters

`ifp`	Pointer to the interface.
`rtp_hdr`	Real-time protocol header.

Output Parameters

None

Return Values

NSM_SUCCESS

nsm_tdm_show_statistics

Use this function to display TDM statistics.

Syntax

```
int
nsm_tdm_show_statistics(struct cli *cli, u_int8_t iwf_type, struct interface *ifp);
```

Input Parameters

`cli`	Specifies the instance of the CLI structure.
`iwf_type`	Tunnel ID.
`ifp`	Interface pointer

Output Parameters

None

© 2015 IP Infusion Inc. Proprietary

Return Values

NSM_SUCCESS

nsm_tdm_set_error_period

Use this function to set the error period.

Syntax

```
int
nsm_tdm_set_error_period (struct interface *ifp, u_int16_t error_time, u_int8_t type2 )
```

Input Parameters

`ifp`	Pointer to the interface
`error_time`	Error time.
`type2`	Type 2.

Output Parameters

None

Return Values

NSM_SUCCESS

© 2015 IP Infusion Inc. Proprietary

CHAPTER 14 Data Structures

This chapter describes the data structures that are used with the MPLS APIs. It includes the following data structure types:

Common Data Structures

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

- `interface`
- `lib_globals`
- `nsm_master`
- `pal_in4_addr`
- `thread`

MPLS Data Structures

The following data structures are used in MPLS.

nsm_mpls

This data structure holds MPLS information. It is defined in the `nsm/mpls/nsm_mpls.h` file.

Member	Description
`iflist`	List of interfaces
`b_lsp`	List of B-LSPs
`min_label_val`	Label pool boundary
`max_label_val`	Label pool boundary
`ingress_ttl`	Custom TTL value
`egress_ttl`	Custom TTL value
`propagate_ttl`	TTL-Propagation capability
`admin_group_array`	Admin groups
`shutdown`	Shut down the session
`kern_msgs`	Kernel messages
`flags`	MPLS flags
`config`	Configured data
`ls_table`	Label space table
`trees`	Tree arrays of pointer to valid trees
`ftn_ix_table4`	Global FTN tables; these tables are also valid for non-packet entries, as the control plane address is added to the FTN table as a dummy entry
`ftn_ix_table6`	
`ftn_pbb_table`	For non-IP based tables, we can use AVL Tree instead of Patricia Tree as Longest Match is not required. The index uses the FTN IX Table Index. However, Patricia Tree is used so that the same structure can be passed along for all FTN Tables.
`ftn_tdm_table`	For structure-agnostic TDM
`lsp_dep_up`	Dependency tables: dependency is always on an IP addressed FTN entry; the tables remain route tables because of this.
`lsp_dep_down`	
`ilm_ix_table`	ILM table

© 2015 IP Infusion Inc. Proprietary

Member	Description
xc_ix_table	XC table
nhlfe_ix_table4	NHLFE tables
nhlfe_ix_table6	
pbb_ilm_ix	Cross connect table
pbb_xc_ix	Holder for tree and index manager
pbb_nhlfe_ix	Holder for tree and index manager
tdm_ilm_ix	For structure-agnostic TDM
tdm_xc_ix	Holder for tree and index manager
tdm_nhlfe_ix	Holder for tree and index manager
vrf_hash	VRF hash table
mapped_routes	Mapped-routes table
mapped_lsp	Mapped-LSP table
mapped_routes_pbb	Mapped routes ptree structure
mapped_lsp_pbb	Mapped LSP Avl_tree structure
mapped_routes_tdm	Mapped routes ptree structure
mapped_lsp_tdm	Mapped LSP Avl_tree structure
vc_table	Virtual circuit routing table structure
vc_group_list	Virtual circuit group data
lsr_id	LDP ID values
index	LDP VC entity index
vpls_table	VPLS table ptree structure
mpls_oam_list	MPLS OAM master data
mpls_oam_read	Thread
oam_sock	Socket
oam_s_sock	Socket
mpls_oam_itut_list	MPLS OAM ITUT data
mpls_vpn_list	MPLS VPN data
label_space	Label space ID

© 2015 IP Infusion Inc. Proprietary

Member	Description
nsm_mpls_circuit_hash_table	Hash table data for MPLS circuit
vc_indx_mgr	Virtual circuit index manager
bfd_fec_conf_list	List of BFD for FEC conf entries
bfd_lsp_conf	BFD attributes, one member for each LSP-type
bfd_flag	BFD flag
lsp_model	LSP model
dscp_exp_map	Configure DSCP to EXP bit mapping
supported_dscp	Supported DSCP values
te_class	TE class table
ct_name	Class type table
stats	Top level stats
if_stats	MPLS Interface stats, per platform
notificationcntl	Flag For Trap Generation: 1 - enabled, 2 - disabled
notify_time_init	Trap variables
notify_cnt	Trap variables
vc_pw_notification	Virtual circuit pseudo wire notification
vc_pw_notification_rate	Virtual circuit pseudo wire notification rate
traps	MPLS SNMP trap callback function
vccv_stats	Place holder for VCCV statistics
loop_gindex	Datalink - loopback index

Definition

```
struct nsm_mpls
{
  /* List of interfaces. */
  struct list *iflist;

  /* List of B-LSP's */
  struct list *b_lsp;

#if (!defined (HAVE_GMPLS) || defined (HAVE_PACKET))
  /* Label pool boundaries. */
  u_int32_t min_label_val;
```


© 2015 IP Infusion Inc. Proprietary

```
  u_int32_t max_label_val;

  /* Custom TTL values. */
#define NSM_MPLS_NO_TTL                -1
  short ingress_ttl;
  short egress_ttl;

  /* TTL-Propagation capability */
  u_char propagate_ttl;

#endif /* !HAVE_GMPLS || HAVE_PACKET */

#ifdef HAVE_TE
  struct admin_group admin_group_array [ADMIN_GROUP_MAX];
#endif /* HAVE_TE */

  u_char shutdown;

  /* Kernel messages. */
#define NSM_MPLS_SHOW_MSG_ERROR        (1 << 0)
#define NSM_MPLS_SHOW_MSG_WARNING      (1 << 1)
#define NSM_MPLS_SHOW_MSG_DEBUG        (1 << 2)
#define NSM_MPLS_SHOW_MSG_NOTICE       (1 << 3)
#define NSM_MPLS_SHOW_MSG_ALL          (1 << 4)
  u_char kern_msgs;

#define NSM_MPLS_FLAG_INSTALL_BK_LSP           (1 << 0)
  u_char flags;

/* Configured data. */
#define NSM_MPLS_CONFIG_INGRESS_TTL    (1 << 0)
#define NSM_MPLS_CONFIG_EGRESS_TTL     (1 << 1)
#define NSM_MPLS_CONFIG_LCL_PKT_HANDLE (1 << 2)
#define NSM_MPLS_CONFIG_KERN_MSGS      (1 << 3)
#define NSM_MPLS_CONFIG_ENABLE_ALL_IFS (1 << 4)
#define NSM_MPLS_CONFIG_MIN_LABEL_VAL  (1 << 5)
#define NSM_MPLS_CONFIG_MAX_LABEL_VAL  (1 << 6)
#define NSM_MPLS_CONFIG_SUPPORTED_DSCP (1 << 7)
#define NSM_MPLS_CONFIG_DSCP_EXP_MAP   (1 << 8)
#ifdef  HAVE_MPLS_TP
#define NSM_MPLS_CONFIG_MPLS_TP_IDENT  (1 << 9)
#endif

  u_int16_t config;

  /* Label space table. */
  struct route_table *ls_table;

  /* Tree arrays of pointer to valid trees */
#define NSM_MPLS_TREE_ARRAY_FTN_TREES            0
```

```
#define NSM_MPLS_TREE_ARRAY_FTN_IX_TREES        1
#define NSM_MPLS_TREE_ARRAY_ILM_TREES           2
#define NSM_MPLS_TREE_ARRAY_ILM_IX_TREES        3
#define NSM_MPLS_TREE_ARRAY_NHLFE_TREES         4
#define NSM_MPLS_TREE_ARRAY_MAPROUTE_TREES      5
#define NSM_MPLS_TREE_ARRAY_FTN6_TREES          6
#define NSM_MPLS_TREE_ARRAY_FTN6_IX_TREES       7
#define NSM_MPLS_TREE_ARRAY_NHLFE6_TREES        8
#define NSM_MPLS_TREE_ARRAY_MAX                 9

  struct nsm_mpls_tree_array trees [NSM_MPLS_TREE_ARRAY_MAX];

  /* Global FTN table. These tables will be valid for non-packet entries too,
     as the control plane address is added to the FTN table as dummy entry */
  struct ptree_ix_table ftn_ix_table4;
#ifdef HAVE_IPV6
  struct ptree_ix_table ftn_ix_table6;
#endif
#ifdef HAVE_GMPLS
  /* For non IP based tables we can use AVL Tree instead of Patricia Tree as
     Longest Match is not required. The index will use the FTN IX Table Index.
     However Patricia Tree is used so that the same structure can be passed
     along for all FTN Tables */
#ifdef HAVE_PBB_TE
  struct ptree_ix_table ftn_pbb_table;
#endif /* HAVE_PBB_TE */

#ifdef HAVE_TDM
  /* For structure agnostic TDM */
  struct ptree_ix_table ftn_tdm_table;
#endif /* HAVE_TDM */
#endif /* HAVE_GMPLS */

#ifdef HAVE_MPLS_P2MP
 /* TE Tunnel 'name' Hash */
 struct hash *te_tnl_hash;

#define NSM_TE_TNL_HASH  NSM_MPLS->te_tnl_hash
#endif /* HAVE_MPLS_P2MP */

  /* Dependency tables - Dependency will always be on an IP addressed FTN entry
     The table stays a route table because of that. */
  struct route_table *lsp_dep_up;
  struct route_table *lsp_dep_down;

  /* ILM table. */
  struct avl_ix_table ilm_ix_table;

  /* XC Table */
  struct avl_ix_table xc_ix_table;
```

 © 2015 IP Infusion Inc. Proprietary

```
  /* NHLFE Table */
  struct avl_ix_table nhlfe_ix_table4;

#ifdef HAVE_IPV6
  struct avl_ix_table nhlfe_ix_table6;
#endif

#ifdef HAVE_GMPLS
  /* Cross connect Table */
#ifdef HAVE_PBB_TE
  struct avl_ix_table pbb_ilm_ix;

  struct avl_ix_table pbb_xc_ix;

  struct avl_ix_table pbb_nhlfe_ix;
#endif /* HAVE_PBB_TE */

#ifdef HAVE_TDM
  /* For structure agnostic TDM */
  struct avl_ix_table tdm_ilm_ix;

  struct avl_ix_table tdm_xc_ix;

  struct avl_ix_table tdm_nhlfe_ix;
#endif /* HAVE_TDM */
#endif /* HAVE_GMPLS */

#ifdef HAVE_VRF
  /* VRF hash table. */
  struct vrf_table *vrf_hash[VRF_HASH_SIZE];
#endif /* HAVE_VRF */

#ifdef HAVE_PACKET
  /* Mapped-routes table. */
  struct ptree *mapped_routes;

  /* Mapped-lsp table. */
  struct avl_tree *mapped_lsp;
#endif /* HAVE_PACKET */

#ifdef HAVE_GMPLS
#ifdef HAVE_PBB_TE
  struct ptree *mapped_routes_pbb;

  struct avl_tree *mapped_lsp_pbb;
#endif /* HAVE_PBB_TE */

#ifdef HAVE_TDM
  struct ptree *mapped_routes_tdm;
```

```
  struct avl_tree *mapped_lsp_tdm;
#endif /* HAVE_TDM */
#endif /* HAVE_GMPLS */

#ifdef HAVE_MPLS_VC
  struct route_table *vc_table;
  struct list *vc_group_list;

  /* ldp id values */
  u_int32_t lsr_id;
  /* ldp VC entity index */
  u_int32_t index;
#endif /* HAVE_MPLS_VC */

#ifdef HAVE_VPLS
  struct ptree *vpls_table;
#endif /* HAVE_VPLS */

#ifdef HAVE_NSM_MPLS_OAM
  /* MPLS OAM Master Data */
  struct list *mpls_oam_list;

  struct thread *mpls_oam_read;
  int oam_sock;
  int oam_s_sock;

  /* MPLS OAM ITUT Data */
  struct list *mpls_oam_itut_list;

#endif /*HAVE_NSM_MPLS_OAM */

#if defined HAVE_MPLS_OAM || defined HAVE_NSM_MPLS_OAM
  struct list *mpls_vpn_list;
#endif /* HAVE_MPLS_OAM || HAVE_NSM_MPLS_OAM */

#ifdef HAVE_MPLS_VC
  u_int16_t label_space;

  struct hash *nsm_mpls_circuit_hash_table;
#define NSM_MPLS_CIRCUIT_HASH NSM_MPLS->nsm_mpls_circuit_hash_table

#ifdef HAVE_MS_PW
  struct hash *nsm_ms_pw_hash_table;
#define NSM_MS_PW_HASH  NSM_MPLS->nsm_ms_pw_hash_table
#define NSM_MS_PW_MIN_IX   1
#define NSM_MS_PW_MAX_IX   65535
#define NSM_MS_PW_BUCKET_SIZE 1280
#endif /* HAVE_MS_PW */
#endif /* HAVE_MPLS_VC */
```

 © 2015 IP Infusion Inc. Proprietary

```
#if (defined HAVE_MPLS_VC) || (defined HAVE_VPLS)
#ifdef HAVE_SNMP
  struct bitmap *vc_indx_mgr;
#endif /*HAVE_SNMP*/
#endif /*defined HAVE_MPLS_VC || defined HAVE_VPLS*/

#if defined (HAVE_BFD) && defined (HAVE_MPLS_OAM)
  struct list *bfd_fec_conf_list; /** List of BFD for FEC conf entries */
   /** BFD attributes, one member for each lsp-type */
  struct nsm_mpls_bfd_lsp_conf bfd_lsp_conf [BFD_MPLS_LSP_TYPE_UNKNOWN];
  u_char bfd_flag; /** BFD Flag. */

  #define NSM_MPLS_BFD_ALL_LDP        (1 << 0)
  #define NSM_MPLS_BFD_ALL_RSVP       (1 << 1)
  #define NSM_MPLS_BFD_ALL_STATIC     (1 << 2)
#endif /* HAVE_BFD  && HAVE_MPLS_OAM */

#if (!defined (HAVE_GMPLS) || defined (HAVE_PACKET))
#ifdef HAVE_DIFFSERV
  /* LSP Model */
  u_char lsp_model;

  /* Configure DSCP to EXP bit mapping. */
  u_char dscp_exp_map[DIFFSERV_MAX_DSCP_EXP_MAPPINGS];

 /* Supported DSCP values. */
  u_char supported_dscp[DIFFSERV_MAX_SUPPORTED_DSCP];

#endif /* HAVE_DIFFSERV */

#ifdef HAVE_DSTE
  /* TE CLASS table */
  struct te_class_s te_class[MAX_TE_CLASS];

  /* CLASS TYPE table */
  char ct_name[MAX_CLASS_TYPE + 1][MAX_CT_NAME_LEN + 1];
#endif /* HAVE_DSTE */
#endif /* !HAVE_GMPLS || HAVE_PACKET */

#if defined (HAVE_MPLS_FWD)
  /* Top level stats. */
  pal_fib_stats_t stats;
  /* MPLS Interface stats. for per platform */
  pal_mpls_if_entry_stats_t if_stats;
#endif /* HAVE_MPLS_FWD */

  /*Flag For Trap Generation */
  int notificationcntl; /* 1 - Enabled, 2 - Disabled */
```

```
  /* Trap variables */
  pal_time_t notify_time_init;
  u_int32_t notify_cnt;

  int vc_pw_notification;
  #define NSM_MPLS_VC_UP_DN_NOTIFN           0x01
  #define NSM_MPLS_VC_DEL_NOTIFN             0x02

  #define NSM_MPLS_VC_UP_DN_NOTIFN_DIS       0
  #define NSM_MPLS_VC_UP_DN_NOTIFN_ENA       1

  #define NSM_MPLS_SNMP_VC_UP_DN_NTFY_ENA    0x01
  #define NSM_MPLS_SNMP_VC_UP_DN_NTFY_DIS    0x02


  #define NSM_MPLS_VC_DEL_NOTIFN_DIS         0
  #define NSM_MPLS_VC_DEL_NOTIFN_ENA         1

  #define NSM_MPLS_SNMP_VC_DEL_NTFY_ENA      0x01
  #define NSM_MPLS_SNMP_VC_DEL_NTFY_DIS      0x02
  int vc_pw_notification_rate;

  /* MPLS SNMP trap callback function */
  vector traps [MPLS_TRAP_ID_MAX];

#ifdef HAVE_VCCV
  /* Place holder for VCCV Statistics */
  struct vccv_statistics vccv_stats;
#endif /* HAVE_VCCV */
#ifdef HAVE_GMPLS
/* Datalink - loopback index */
  s_int32_t loop_gindex;
#endif /* HAVE_GMPLS */

#ifdef HAVE_MPLS_TP
 /* Placeholder for GlobalId , NodeId for this node */
 struct fec_entry_mpls_tp  local_id;

 /* Tunnel 'name' Hash */
 struct hash *tnl_hash;

 /*Tunnel Table (Ingress) */
 struct avl_tree *ing_tnl_tbl;

 /*Tunnel Table (transit) */
 struct avl_tree *tran_tnl_tbl;

/*Tunnel Table (Egress) */
 struct avl_tree *egr_tnl_tbl;
```


© 2015 IP Infusion Inc. Proprietary

```
 /*MPLS-TP NHLFE Table */
 struct avl_ix_table nhlfe_mpls_tp_ix_tbl;

 /* Tunnel Index Generator */
 struct bitmap *tnl_ix_mgr;

 /* MPLS-TP Tunnel IPV4 mapped route Dependancy table */
 struct ptree *mpls_tp_map_ipv4_tbl;

  struct avl_tree *mpls_tp_tnl_cache_tree;
  struct avl_tree *mpls_tp_vc_cache_tree;
  struct avl_tree *mpls_tp_if_cache_tree;

#endif /*HAVE_MPLS_TP */

struct avl_tree *bw_class_tbl;

#ifdef HAVE_HA
 s_int32_t remove_stale_proto_id;
 bool_t remove_stale_ilm_force_del;
 HA_CDR_REF remove_stale_cdr_ref;
#endif /* HAVE_HA */
};
```

fec_gen_entry

This data structure holds information about an forwarding equivalency class (FEC). It is defined in the `lib/mpls/mpls.h` file.

Member	Description
`type`	GMPLS entry type
`prefix`	FEC prefix
`tdm`	FEC entry TDM (physical interface identifier and time slot identifier)
`pbb`	Provider backbone
`u`	Union of all FTN types

Definition

```
struct fec_gen_entry
{
  enum gmpls_entry_type type;

  /* Union of all FTN types */
  union {
    struct prefix prefix;
#ifdef HAVE_GMPLS
```

```
#ifdef HAVE_TDM
    struct fec_entry_tdm tdm;
#endif /* HAVE_TDM */
#ifdef HAVE_PBB_TE
    struct fec_entry_pbb_te pbb;
#endif /* HAVE_PBB_TE */
#ifdef HAVE_MPLS_TP
    struct fec_entry_mpls_tp tp;
#endif /* HAVE_MPLS_TP */
#endif /* HAVE_GMPLS */
  } u;
};
```

MPLS Pseudowire Data Structure

Data structures used in MPLS Pseudowire are defined in this subsection.

ldp_id

This data structure holds an LDP identifier. It is defined in the `ldpd/ldpd.h` file.

Member	Description
`lsr_id`	Label switched router ID
`label_space`	Label space ID

Description

```
struct ldp_id
{
  u_int32_t lsr_id;
  u_int16_t label_space;
};
```

MPLS-NSF Data Structures

Data structures used in MPLS-NSF (Nonstop Forwarding) are defined in this subsection.

nsm_mpls_ftn_msg_key_t

This data structure is defined in the `nsm/mpls/ha/nsm_mpls_cal.h` file.

© 2015 IP Infusion Inc. Proprietary

Member	Description
vr_id	VR identifier
ftn_ix	FEC-to-NHLFE index

Definition

```
typedef struct nsm_mpls_ftn_msg_key_t
{
  u_int32_t vr_id;
  u_int32_t ftn_ix;
} NSM_MPLS_FTN_MSG_KEY_T;
```

nsm_mpls_ftn_msg_data_t

This data structure is defined in the nsm/mpls/ha/nsm_mpls_cal.h file.

Member	Description
ha_data	High availability data
f_type	FTN entry flag
config	Configuration

Definition

```
typedef struct nsm_mpls_ftn_msg_data_t
{
  struct ftn_add_gen_data ha_data;
  mpls_ftn_type_t f_type;
  bool_t config;
} NSM_MPLS_FTN_MSG_DATA_T;
```

nsm_mpls_ilm_msg_key_t

This data structure is defined in the nsm/mpls/ha/nsm_mpls_cal.h file.

Member	Description
vr_id	VR identifier
ilm_ix	ILM index
iif_ix	Interface index

Definition

```
typedef struct nsm_mpls_ilm_msg_key_t
```

```
{
  u_int32_t vr_id;
  u_int32_t ilm_ix;
  u_int32_t iif_ix;
} NSM_MPLS_ILM_MSG_KEY_T;
```

nsm_mpls_ilm_msg_data_t

This data structure is defined in the `nsm/mpls/ha/nsm_mpls_cal.h` file.

Member	Description
`ha_data`	High availability data
`mapped_ilm`	Mapped ILM
`config`	Configuration

Definition

```
typedef struct nsm_mpls_ilm_msg_data_t
{
  struct ilm_add_gen_data ha_data;
  bool_t mapped_ilm;
  bool_t config;
} NSM_MPLS_ILM_MSG_DATA_T;
```

nsm_mpls_if_key_t

This data structure is defined in the `nsm/mpls/ha/nsm_mpls_cal.h` file.

Member	Description
`ifindex`	Interface index

Definition

```
typedef struct nsm_mpls_if_key_t
{
  u_int32_t ifindex;
} NSM_MPLS_IF_KEY_T;
```

nsm_mpls_if_data_t

This data structure is defined in the `nsm/mpls/ha/nsm_mpls_cal.h` file.

© 2015 IP Infusion Inc. Proprietary

Member	Description
chkpt_info	Checkpoint information

Definition

```
typedef struct nsm_mpls_ilm_msg_data_t

typedef struct nsm_mpls_if_data_t
{
  s_int32_t chkpt_info;
} NSM_MPLS_IF_DATA_T;
```

nsm_mpls_route_map_cli_key_t

This data structure is defined in the nsm/mpls/ha/nsm_mpls_cal.h file.

Member	Description
vr_id	VR identifier
ftn_ix	FTN index

Definition

```
typedef struct nsm_mpls_route_map_cli_key_t
{
  u_int32_t vr_id;
  u_int32_t ftn_ix;
} NSM_MPLS_ROUTE_MAP_CLI_KEY_T;
```

nsm_mpls_route_map_cli_key_t

This data structure is defined in the nsm/mpls/ha/nsm_mpls_cal.h file.

Member	Description
ent	Entry
fec	Prefix FEC

Definition

```
typedef struct nsm_mpls_route_map_cli_data_t
{
  struct fec_gen_entry ent;
  struct prefix fec;
} NSM_MPLS_ROUTE_MAP_CLI_DATA_T;
```

nsm_mpls_remove_stale_key_t

This data structure is defined in the `nsm/mpls/ha/nsm_mpls_cal.h` file.

Member	Description
`vr_id`	VR identifier

Definition

```
typedef struct nsm_mpls_remove_stale_key_t
{
  u_int32_t vr_id;
}NSM_MPLS_REMOVE_STALE_KEY_T;
```

nsm_mpls_remove_stale_data_t

This data structure is defined in the `nsm/mpls/ha/nsm_mpls_cal.h` file.

Member	Description
`proto_id`	Protocol identifier
`ilm_force_del`	ILM force delete

Definition

```
typedef struct nsm_mpls_remove_stale_data_t
{
  s_int32_t proto_id;
  bool_t ilm_force_del;
}NSM_MPLS_REMOVE_STALE_DATA_T;
```

nsm_mpls_lbl_pool_msg_key_t

This data structure is defined in the `nsm/mpls/ha/nsm_mpls_cal.h` file.

Member	Description
`vr_id`	VR identifier
`label_block`	Label block
`label_space`	Label space
`lbl_format`	Label format
`range_owner`	Range owner

 © 2015 IP Infusion Inc. Proprietary

Definition

```
typedef struct nsm_mpls_lbl_pool_msg_key_t
{
  u_int32_t vr_id;
  u_int32_t label_block;
  u_int16_t label_space;
  enum gmpls_label_type  lbl_format;
  u_int8_t range_owner;
} NSM_MPLS_LBL_POOL_MSG_KEY_T;
```

nsm_mpls_lbl_pool_msg_key_t

This data structure is defined in the `nsm/mpls/ha/nsm_mpls_cal.h` file.

Member	Description
`protocol`	Protocol identifier
`status`	Status

Definition

```
typedef struct nsm_mpls_lbl_pool_msg_data_t
{
  u_int8_t protocol;
  u_char status;
} NSM_MPLS_LBL_POOL_MSG_DATA_T;
```

nsm_mpls_if_status_msg_key_t

This data structure is defined in the `nsm/mpls/ha/nsm_mpls_cal.h` file.

Member	Description
`protocol`	VR identifier
`ifindex`	Interface index

Definition

```
typedef struct nsm_mpls_if_status_msg_key_t
{
  u_int32_t vr_id;
  u_int32_t ifindex;
} NSM_MPLS_IF_STATUS_MSG_KEY_T;
```

nsm_mpls_if_status_msg_data_t

This data structure is defined in the `nsm/mpls/ha/nsm_mpls_cal.h` file.

Member	Description
label_space	Label space

Definition

```
typedef struct nsm_mpls_if_status_msg_data_t
{
  u_int16_t label_space;
} NSM_MPLS_IF_STATUS_MSG_DATA_T;
```

nsm_protocol_restart_state_key_t

This data structure is defined in the nsm/mpls/ha/nsm_mpls_cal.h file.

Member	Description
restart_proto_vr_id	Restart protocol Virtual Router identifier

Definition

```
typedef struct nsm_protocol_restart_state_key_t
{
  u_int32_t restart_proto_vr_id;

  /* Restarting protocol ID. */
  int restart_proto_id;
} NSM_PROTOCOL_RESTART_STATE_KEY_T;
```

nsm_protocol_restart_state_data_t

This data structure is defined in the nsm/mpls/ha/nsm_mpls_cal.h file.

Member	Description
restart_state	Restart state

Definition

```
typedef struct nsm_protocol_restart_state_data_t
{
  /* Restart State */
  u_char restart_state;
} NSM_PROTOCOL_RESTART_STATE_DATA_T;
```


© 2015 IP Infusion Inc. Proprietary

mpls_tp_lps_group

This structure holds information for processing PSC groups. It is defined in the `nsm/mplstp/oam_mplstp.h`, h file.

Member	Description
`group_name`	PSC group name
`mpls_lps`	MPLS Link Protection Switching
`revertive_mode`	Revertive mode setting
`protection_scheme`	Protection scheme setting
`mode`	LPS mode
`wtr_timer`	Wait to restore timer value
`hold_off_timer`	Hold-off timer value
`rapid_tx_freq`	Transmit frequency for Rapid PSC packet
`continual_tx_freq`	Transmit frequency for Normal PSC packet
`start_time`	PSC group start time
`current_state`	Current state of the PSC group
`last_state`	Last state of the PSC group
`farend_state`	State of the PSC group at the far end
`current_event`	Current event on the PSC
`hold_off_trigger_event`	Event that triggered the hold-off timer
`flags`	Flags
`config_flags`	Configuration flags
`mode_mismatch_count`	Count of mismatched modes
`oper_status`	Operational status
`primary_me`	Primary Maintenance Entity
`backup_me`	Backup Maintenance Entity
`lps_counter`	Counter for LPS
`msg_field`	Message
`last_msg_sent`	Last message sent
`msg_recvd`	Message received

© 2015 IP Infusion Inc. Proprietary

Member	Description
lps_pkt_stats	LPS packet statistics
t_wtr	Wait-to-restore thread
t_holdoff	Hold-off timer thread
t_poll_cntr	Polling-counter thread
t_update	Update thread

Definition

```
struct mpls_tp_lps_group
{
  /* PSC Group Name */
  char group_name[PSC_NAME_LENGTH_MAX];

  struct mpls_lps *mpls_lps;

  rever_mode revertive_mode;

  /* unidirectional / bidirectional */
  lps_scheme protection_scheme;

  /* 1+1 / 1:1 / 1:n */
  lps_mode mode;

  /* Wait To Restore Timer Value */
  u_int32_t  wtr_timer;

  /* Hold off timer value */
  u_int32_t hold_off_timer;

  /* Tx frequencey for Rapid PSC packet */
  u_int32_t rapid_tx_freq;

  /* Tx frequencey for Normal PSC packet */
  u_int32_t continual_tx_freq;

  /* PSC Group start time. */
  pal_time_t start_time;

  /* Current state of this PSC Group */
  lps_FSM_state current_state;

  /* Last state of this PSC Group */
  lps_FSM_state last_state;

  /* Last far end state of this PSC Group */
```

 © 2015 IP Infusion Inc. Proprietary

```
  lps_FSM_state farend_state;
 /* Current Event on this PSC Group */
  lps_FSM_event current_event;

  /* Event triggering the hold-off timer */
  lps_FSM_event hold_off_trigger_event;

  /* Defined in lps_fsm.h file */
  u_int32_t flags;

#define PSC_LOCKOUT                     (1 << 0)
#define PSC_SWITCHOVER_MANUAL           (1 << 1)
#define PSC_SWITCHOVER_FORCE            (1 << 2)
  u_int32_t config_flags;

#define PSC_SCHEME_CONFIG_INCOMPLETE       (1 << 0)
#define PSC_PRI_MEG_CONFIG_INCOMPLETE      (1 << 1)
#define PSC_BKP_MEG_CONFIG_INCOMPLETE      (1 << 2)
#define PSC_PRI_REV_MEG_CONFIG_INCOMPLETE  (1 << 3)
#define PSC_BKP_REV_MEG_CONFIG_INCOMPLETE  (1 << 4)
  u_int8_t config_incomp_flag;

  bool_t local_state_change;

  u_int32_t mismatch_flag;

  u_int32_t mode_mismatch_count;

  operational_status oper_status;

  s_int32_t reason_code;

  struct me_entry *primary_me;

  struct me_entry *backup_me;

  /* This will be used in case of assoc tunnel */
  struct me_entry *primary_rev_me;

  struct me_entry *backup_rev_me;
 struct lps_msg_field msg_field;
  struct lps_msg_field last_msg_sent;
  struct lps_msg_field msg_recvd;
  struct lps_pkt_statistics lps_pkt_stats;
  struct thread *t_wtr;
  struct thread *t_holdoff;
  /* Timer for polling the interface card for counter reads */
  struct thread *t_poll_cntr;
};
```

MPLS-TP Data Structures

The following data structures are used in fault management, delay and loss measurement, loopback and lock instruct processing for MPLS-TP.

mplstp_tnl_params

This definition contains information about MPLS-TP tunnels. It is defined in the `lib/oam_mpls_msg.h` file.

Member	Description
`role`	Tunnel role
`mode`	Tunnel mode
`tnl_id`	Tunnel ID
`lsp_id`	LSP ID
`src`	Source FEC entry
`dst`	Destination FEC entry
`rev_tnl_id`	Reverse tunnel ID
`rev_lsp_id`	Reverse tunnel LSP
`tnl_status`	Tunnel status
`rev_meg_index`	Reverse MEG index
`rev_me_index`	Reverse ME index
`fwd`	MPLS-TP forward label information
`rev`	MPLS-TP reverse label information
`tnl_name_length`	Length of tunnel name
`tnl_name`	Tunnel name

Definition

```
struct mplstp_tnl_params
{
  u_int32_t  role;
  u_int32_t  mode;
  u_int16_t  tnl_id;
  u_int16_t  lsp_id;
  struct fec_entry_mpls_tp src;
  struct fec_entry_mpls_tp dst;
  u_int16_t  rev_tnl_id;
  u_int16_t  rev_lsp_id;
```

 © 2015 IP Infusion Inc. Proprietary

```
  u_int32_t  tnl_status;
  u_int32_t rev_meg_index;
  u_int32_t rev_me_index;
  struct mplstp_label_info fwd;
  struct mplstp_label_info rev;
  u_int32_t tnl_name_len;
  u_char* tnl_name;
};
```

mplstp_vc_params

This definition contains information about MPLS-TP virtual circuits. It is defined in the `lib/oam_mpls_msg.h` file.

Member	Description
`vc_id`	Virtual circuit ID
`agi_length`	Attachment Group Identifier length
`agi`	Attachment Group Identifier
`src`	Source FEC entry
`scr_ac_id`	Source attachment circuit identifier
`dst`	Destination FEC entry
`dst_ac_id`	Destination attachment circuit identifier
`out_if`	Out interface
`tnl_id`	Tunnel ID
`ac_if_index`	Attachment circuit interface index
`out_vc_label`	VC label at egress
`in_vc_label`	VC label at ingress
`cc_type`	Continuity checking type
`cv_type`	Connectivity validation type
`vc_status`	VC status
`vlan_id`	ID of the VLAN

Definition

```
struct mplstp_vc_params
{
  u_int32_t vc_id;
  u_int8_t  agi_len;
  u_char *agi;
```

```
  struct fec_entry_mpls_tp src;
  u_int32_t src_ac_id;
  struct fec_entry_mpls_tp dst;
  u_int32_t dst_ac_id;
  u_int32_t out_if;
  u_int16_t tnl_id;
  u_int32_t ac_if_index;
  u_int32_t out_vc_label;
  u_int32_t in_vc_label;
  u_int8_t  cc_type; /** CC Type */
  u_int8_t  cv_type; /** CV Type */
  u_int32_t vc_status;
  u_int32_t vlan_id;
};
```

MPLS-TP LPS Data Structures

The data structure used in MPLS-TP linear protection switching (LPS) is defined in this subsection.

mpls_tp_lps_group

This structure holds information for processing PSC groups. It is defined in the `nsm/mplstp/oam_mplstp.h` file.

Member	Description
`group_name`	PSC group name
`mpls_lps`	MPLS Link Protection Switching
`revertive_mode`	Revertive mode setting
`protection_scheme`	Protection scheme setting
`mode`	LPS mode
`wtr_timer`	Wait to restore timer value
`hold_off_timer`	Hold-off timer value
`rapid_tx_freq`	Transmit frequency for Rapid PSC packet
`continual_tx_freq`	Transmit frequency for Normal PSC packet
`start_time`	PSC group start time
`current_state`	Current state of the PSC group
`last_state`	Last state of the PSC group
`farend_state`	State of the PSC group at the far end
`current_event`	Current event on the PSC

 © 2015 IP Infusion Inc. Proprietary

Member	Description
hold_off_trigger_event	Event that triggered the hold-off timer
flags	Flags
config_flags	Configuration flags
mode_mismatch_count	Count of mismatched modes
oper_status	Operational status
primary_me	Primary Maintenance Entity
backup_me	Backup Maintenance Entity
lps_counter	Counter for LPS
msg_field	Message
last_msg_sent	Last message sent
msg_recvd	Message received
lps_pkt_stats	LPS packet statistics
t_wtr	Wait-to-restore thread
t_holdoff	Hold-off timer thread
t_poll_cntr	Polling-counter thread
t_update	Update thread

Definition

```
struct mpls_tp_lps_group
{
  /* PSC Group Name */
  char group_name[PSC_NAME_LENGTH_MAX];

  struct mpls_lps *mpls_lps;

  rever_mode revertive_mode;

  /* unidirectional / bidirectional */
  lps_scheme protection_scheme;

  /* 1+1 / 1:1 / 1:n */
  lps_mode mode;

  /* Wait To Restore Timer Value */
  u_int32_t  wtr_timer;

  /* Hold off timer value */
```

```
  u_int32_t hold_off_timer;

  /* Tx frequencey for Rapid PSC packet */
  u_int32_t rapid_tx_freq;

  /* Tx frequencey for Normal PSC packet */
  u_int32_t continual_tx_freq;

  /* PSC Group start time. */
  pal_time_t start_time;

  /* Current state of this PSC Group */
  lps_FSM_state current_state;

  /* Last state of this PSC Group */
  lps_FSM_state last_state;

  /* Last far end state of this PSC Group */
  lps_FSM_state farend_state;

  /* Current Event on this PSC Group */
  lps_FSM_event current_event;
 /* Event triggering the hold-off timer */
  lps_FSM_event hold_off_trigger_event;

  /* Defined in lps_fsm.h file */
  u_int32_t flags;

#define PSC_LOCKOUT                     (1 << 0)
#define PSC_SWITCHOVER_MANUAL           (1 << 1)
#define PSC_SWITCHOVER_FORCE            (1 << 2)
  u_int32_t config_flags;

#define PSC_SCHEME_CONFIG_INCOMPLETE       (1 << 0)
#define PSC_PRI_MEG_CONFIG_INCOMPLETE      (1 << 1)
#define PSC_BKP_MEG_CONFIG_INCOMPLETE      (1 << 2)
#define PSC_PRI_REV_MEG_CONFIG_INCOMPLETE  (1 << 3)
#define PSC_BKP_REV_MEG_CONFIG_INCOMPLETE  (1 << 4)
  u_int8_t config_incomp_flag;

  bool_t local_state_change;

  u_int32_t mismatch_flag;

  u_int32_t mode_mismatch_count;

  operational_status oper_status;

  s_int32_t reason_code;
```

 © 2015 IP Infusion Inc. Proprietary

```
  struct me_entry *primary_me;

  struct me_entry *backup_me;

  /* This will be used in case of assoc tunnel */
  struct me_entry *primary_rev_me;

  struct me_entry *backup_rev_me;

 struct lps_msg_field msg_field;
  struct lps_msg_field last_msg_sent;
  struct lps_msg_field msg_recvd;
  struct lps_pkt_statistics lps_pkt_stats;
  struct thread *t_wtr;
  struct thread *t_holdoff;
  /* Timer for polling the interface card for counter reads */
  struct thread *t_poll_cntr;
};
```

© 2015 IP Infusion Inc. Proprietary

CHAPTER 15 MPLS MIB API

This chapter describes Management Information Base (MIB) support for Multi-Protocol Label Switching (MPLS).

Supported Tables

The MPLS MIB is implemented in accordance with these standards:

- RFC 3813: MPLS Label Switching Router (LSR) MIB
- RFC 3814: MPLS Forwarding Equivalence Class To Next Hop Label Forwarding Entry (FEC-To-NHLFE) MIB

The MIB tables and variables listed in the sections below are supported in ZebOS-XP. The tables contain cross-references to the function for each Object Type.

mplsFTNTable

This table, defined in RFC 3814, defines FEC to NHLFE mappings.

Object Type	Syntax	Access	Functions
mplsFTNRowStatus	RowStatus	read-create	nsm_gmpls_set_ftn_row_status
mplsFTNActionPointer	RowPointer	read-create	nsm_mpls_set_ftn_act_pointer
mplsFTNActionType	INTEGER	read-create	nsm_mpls_set_ftn_act_type
mplsFTNAddrType	InetAddressType	read-create	nsm_mpls_set_ftn_addr_type
mplsFTNDescr	SnmpAdminString	read-create	nsm_mpls_set_ftn_descr
mplsFTNDscp	Dscp	read-create	nsm_mpls_set_ftn_dscp
mplsFTNDestAddrMax mplsFTNDestAddrMin	InetAddress InetAddress	read-create read-create	nsm_mpls_set_ftn_dst_addr_min_max
mplsFTNMask	BITS	read-create	nsm_mpls_set_ftn_mask
mplsFTNProtocol	Integer32	read-create	nsm_mpls_set_ftn_protocol
mplsFTNSourceAddrMax mplsFTNSourceAddrMin	InetAddress InetAddress	read-create read-create	nsm_mpls_set_ftn_src_addr_min_max
mplsFTNDestPortMax mplsFTNSourcePortMax	InetPortNumber InetPortNumber	read-create read-create	nsm_mpls_set_ftn_src_dst_port_max
mplsFTNDestPortMin mplsFTNSourcePortMin	InetPortNumber InetPortNumber	read-create read-create	nsm_mpls_set_ftn_src_dst_port_min
mplsFTNStorageType	StorageType	read-create	nsm_mpls_set_ftn_st_type

mplsInSegmentTable

This table, defined in RFC 3813, represents a collection of incoming segments to an LSR.

Object Type	Syntax	Access	Functions
mplsInSegmentLabelPtr	RowPointer	read-create	nsm_mpls_set_inseg_lb_ptr
mplsInSegmentNPop	Integer32	read-create	nsm_mpls_set_inseg_npop
mplsInSegmentAddrFamily	AddressFamilyNumbers	read-create	nsm_mpls_set_inseg_addr_family
mplsInSegmentTrafficParamPtr	RowPointer	read-create	nsm_mpls_set_inseg_tf_prm
mplsInSegmentRowStatus	RowStatus	read-create	nsm_mpls_set_inseg_row_status
mplsInSegmentStorageType	StorageType	read-create	nsm_mpls_set_inseg_st_type
mplsInSegmentInterface	InterfaceIndex	read-create	nsm_mpls_set_inseg_if
mplsInSegmentTrafficParamPtr	RowPointer	read-create	nsm_mpls_set_inseg_tf_prm

mplsOutSegmentTable

This table, defined in RFC 3813, represents a collection of outcoming segments from an LSR.

Object Type	Syntax	Access	Functions
mplsOutSegmentIfIndex	InterfaceIndex	read-create	nsm_mpls_set_outseg_if_ix
mplsOutSegmentNextHopIpv4Addr	InetAddressIPv4	read-create	nsm_mpls_set_outseg_nxt_hop_ipa
mplsOutSegmentNextHopIpAddrType	InetAddressType	read-create	nsm_mpls_set_outseg_nxt_hop_ipa_type
mplsOutSegmentPushTopLabel	TruthValue	read-create	nsm_mpls_set_outseg_push_top_lb
mplsOutSegmentRowStatus	RowStatus	read-create	nsm_mpls_set_outseg_row_status
mplsOutSegmentStorageType	StorageType	read-create	nsm_mpls_set_outseg_st_type
mplsOutSegmentTrafficParamPtr	RowPointer	read-create	nsm_mpls_set_outseg_tf_prm
mplsOutSegmentTopLabel	MplsLabel	read-create	nsm_mpls_set_outseg_top_lb
mplsOutSegmentTopLabelPtr	RowPointer	read-create	nsm_mpls_set_outseg_top_lb_ptr
mplsOutSegmentInterface	InterfaceIndex	read-create	nsm_mpls_set_outseg_if_ix
mplsOutSegmentNextHopAddrType	InetAddressType	read-create	nsm_mpls_set_outseg_nxt_hop_ipa_type
mplsOutSegmentNextHopAddr	InetAddressIPv4	read-create	nsm_mpls_set_outseg_nxt_hop_ipa

© 2015 IP Infusion Inc. Proprietary

mplsXCTable

This table, defined in RFC 3813, defines cross-connect entries for switching between LSP segments.

Object Type	Syntax	Access	Functions
mplsXCAdminStatus	INTEGER	read-create	nsm_mpls_set_xc_adm_status
mplsXCLabelStackIndex	Integer32	read-create	nsm_mpls_set_xc_lb_stk_ix
mplsXCLspId	MplsLSPID	read-create	nsm_mpls_set_xc_lspid
mplsXCRowStatus	RowStatus	read-create	nsm_mpls_set_xc_row_status
mplsXCStorageType	StorageType	read-create	nsm_mpls_set_xc_st_type

Notification Configuration

This variable, defined in RFC 3813, enables or disables mplsXCUp and mplsXCDown notifications.

Object Type	Syntax	Access	Functions
mplsXCNotificationsEnable	TruthValue	read-write	nsm_mpls_set_notify

APIs

Function	Description
nsm_gmpls_set_ftn_row_status	Sets the row status for the FTN
nsm_mpls_set_ftn_act_pointer	Sets a pointer to a cross connect entry indicating the LSP to redirect matching packets
nsm_mpls_set_ftn_act_type	Sets the type of action to be taken on packets matching this FTN
nsm_mpls_set_ftn_addr_type	Sets the type of source and destination address
nsm_mpls_set_ftn_descr	Sets the description of this FTN
nsm_mpls_set_ftn_dscp	Sets the DSCP of this FTN
nsm_mpls_set_ftn_dst_addr_min_max	Sets the lower and upper end of the destination address range for this FTN
nsm_mpls_set_ftn_mask	Sets the bitmap that indicates which fields are active for this FTN
nsm_mpls_set_ftn_protocol	Sets the IP protocol to match against the IPv4 protocol number in the packet
nsm_mpls_set_ftn_src_addr_min_max	Sets the lower and upper end of the source address range
nsm_mpls_set_ftn_src_dst_port_max	Sets the upper end of the source and destination port range

Function	Description
nsm_mpls_set_ftn_src_dst_port_min	Sets the lower end of the source and destination port range
nsm_mpls_set_ftn_st_type	Sets the storage type for this FTN
nsm_mpls_set_inseg_addr_family	Sets the address family. for this insegment
NSM_API_SET_ERROR when the function fails	Sets the interface index for the given insegment
nsm_mpls_set_inseg_lb	Sets the label for the given insegment
nsm_mpls_set_inseg_lb_ptr	Sets the label pointer for the given insegment
nsm_mpls_set_inseg_npop	Sets the number of labels to pop from the incoming packet
nsm_mpls_set_inseg_row_status	Sets the row status of the given insegment
nsm_mpls_set_inseg_st_type	Sets the storage type of the given insegment
nsm_mpls_set_inseg_tf_prm	Sets the traffic parameter pointer for the given insegment
nsm_mpls_set_notify	Sets the notification enable value
nsm_mpls_set_outseg_if_ix	Sets the interface index of the outsegment
nsm_mpls_set_outseg_nxt_hop_ipa	Sets the IP address of the next hop for the given outsegment
nsm_mpls_set_outseg_nxt_hop_ipa_type	Sets the next hop IP address type for the given outsegment
nsm_mpls_set_outseg_push_top_lb	Sets whether to push a top label onto the outgoing packets label stack. for the given outsegment
nsm_mpls_set_outseg_row_status	Sets the row status for the given outsegment
nsm_mpls_set_outseg_st_type	Sets the storage type for the given outsegment
nsm_mpls_set_outseg_tf_prm	Sets the traffic parameters for the given outsegment
nsm_mpls_set_outseg_top_lb	Sets the outlabel to push onto the top of the outgoing packets label stack for the given outsegment
nsm_mpls_set_outseg_top_lb_ptr	Sets the label pointer for the given outsegment
nsm_mpls_set_xc_adm_status	Sets the operational status of the given cross connect
nsm_mpls_set_xc_lb_stk_ix	Sets the primary index that identifies a stack of labels to push below the top label
nsm_mpls_set_xc_lspid	Sets the label switched path for the given cross connect
nsm_mpls_set_xc_row_status	Sets the row status of the given cross connect
nsm_mpls_set_xc_st_type	Sets the storage type of the given cross connect

Include Files

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

- `nsm/mpls/nsm_mpls_api.h`

© 2015 IP Infusion Inc. Proprietary

- nsm/mpls/nsm_mpls_snmp.h

nsm_gmpls_set_ftn_row_status

This function sets the row status for the FTN.

Syntax

```
u_int32_t
nsm_gmpls_set_ftn_row_status (struct nsm_master *nm,
                              u_int32_t ftn_ix, u_int32_t row_status)
```

Input Parameters

nm	NSM master
ftn_ix	Index for a conceptual row in the FTN table
row_status	Row status of the FTN entry. These constants are from the mpls_row_status enum in lib/mpls/mpls.h:

RS_ACTIVE

RS_CREATE_GO

RS_CREATE_WAIT

RS_DESTROY

RS_NOT_IN_SERVICE

RS_NOT_READY

Output Parameters

None

Return Values

NSM_API_SET_SUCCESS when the function succeeds

NSM_API_SET_ERROR when the function fails

nsm_mpls_set_ftn_act_pointer

This function sets a pointer to a cross connect entry indicating the LSP to redirect matching packets.

Syntax

```
u_int32_t
nsm_mpls_set_ftn_act_pointer (struct nsm_master *nm, u_int32_t ftn_ix,
                              u_int32_t xc_ix, u_int32_t ilm_ix,
                              u_int32_t nhlfe_ix)
```

Input Parameters

nm	NSM master
ftn_ix	Index to a conceptual row in the FTN table
xc_ix	Cross connect index
ilm_ix	ILM index

`nhlfe_ix`	NHLFE index

Output Parameters

None

Return Values

NSM_API_SET_SUCCESS when the function succeeds

NSM_API_SET_ERROR when the function fails

nsm_mpls_set_ftn_act_type

This function sets the type of action to be taken on packets matching this FTN.

Syntax

```
u_int32_t
nsm_mpls_set_ftn_act_type (struct nsm_master *nm,
                           u_int32_t ftn_ix, u_int32_t act_type)
```

Input Parameters

`nm`	NSM master
`ftn_ix`	Index to a conceptual row in the FTN table
`act_type`	Type of action to take on packets matching this FTN entry; the only permitted value is REDIRECT_LSP

Output Parameters

None

Return Values

NSM_API_SET_SUCCESS when the function succeeds

NSM_API_SET_ERROR when the function fails

nsm_mpls_set_ftn_addr_type

This function sets the type of source and destination address.

Syntax

```
u_int32_t
nsm_mpls_set_ftn_addr_type (struct nsm_master *nm,
                            u_int32_t ftn_ix, u_int32_t addr_type)
```

Input Parameters

`nm`	NSM master
`ftn_ix`	Index to a conceptual row in the FTN table
`addr_type`	Type of source and destination address; the only permitted value is INET_AD_IPV4

© 2015 IP Infusion Inc. Proprietary

Output Parameters

None

Return Values

NSM_API_SET_SUCCESS when the function succeeds

NSM_API_SET_ERROR when the function fails

nsm_mpls_set_ftn_descr

This function sets the description of this FTN.

Syntax

```
u_int32_t
nsm_mpls_set_ftn_descr (struct nsm_master *nm,
                        u_int32_t ftn_ix, char *descr)
```

Input Parameters

nm	NSM master
ftn_ix	Index to a conceptual row in the FTN table
descr	Description of this FTN entry

Output Parameters

None

Return Values

NSM_API_SET_SUCCESS when the function succeeds

NSM_API_SET_ERROR when the function fails

nsm_mpls_set_ftn_dscp

This function sets the DSCP of this FTN.

Syntax

```
u_int32_t
nsm_mpls_set_ftn_dscp (struct nsm_master *nm,
                        u_int32_t ftn_ix, u_int32_t dscp)
```

Input Parameters

nm	NSM master
ftn_ix	Index to a conceptual row in the FTN table
dscp	DSCP field

Output Parameters

None

Return Values

NSM_API_SET_SUCCESS when the function succeeds

NSM_API_SET_ERROR when the function fails

nsm_mpls_set_ftn_dst_addr_min_max

This function sets the lower and upper end of the destination address range for this FTN.

Syntax

```
u_int32_t
nsm_mpls_set_ftn_dst_addr_min_max (struct nsm_master *nm, u_int32_t ftn_ix,
                                   struct pal_in4_addr *addr)
```

Input Parameters

`nm`	NSM master
`ftn_ix`	Index to a conceptual row in the FTN table
`addr`	Lower end of the destination address and upper end of the destination address range

Output Parameters

None

Return Values

NSM_API_SET_SUCCESS when the function succeeds

NSM_API_SET_ERROR when the function fails

nsm_mpls_set_ftn_mask

This function sets the bitmap that indicates which fields (source address range, destination address range, source port range, destination port range, IPv4 protocol field, and DSCP) are active for this FTN.

Syntax

```
u_int32_t
nsm_mpls_set_ftn_mask (struct nsm_master *nm,
                       u_int32_t ftn_ix, u_int32_t ftn_mask)
```

Input Parameters

`nm`	NSM master
`ftn_ix`	Index to a conceptual row in the FTN table
`ftn_mask`	Bit map indicating which fields (source address range, destination address range, source port range, destination port range, ipv4 protocol field, and DSCP) are active for this FTN entry; the only permitted value is NSM_MPLS_FTN_MASK_DESTADDRESS.

Output Parameters

None

© 2015 IP Infusion Inc. Proprietary

Return Values

NSM_API_SET_SUCCESS when the function succeeds

NSM_API_SET_ERROR when the function fails

nsm_mpls_set_ftn_protocol

This function sets the IP protocol to match against the IPv4 protocol number in the packet.

Syntax

```
u_int32_t
nsm_mpls_set_ftn_protocol (struct nsm_master *nm,
                           u_int32_t ftn_ix, u_int32_t protocol)
```

Input Parameters

nm	NSM master
ftn_ix	Index to a conceptual row in the FTN table
protocol	IP protocol to match against the IPv4 protocol number in the packet; the only permitted value is match all (255)

Output Parameters

None

Return Values

NSM_API_SET_SUCCESS when the function succeeds

NSM_API_SET_ERROR when the function fails

nsm_mpls_set_ftn_src_addr_min_max

This function sets the lower and upper end of the source address range.

Syntax

```
u_int32_t
nsm_mpls_set_ftn_src_addr_min_max (struct nsm_master *nm, u_int32_t ftn_ix,
                                   struct pal_in4_addr *addr)
```

Input Parameters

nm	NSM master
ftn_ix	Index to a conceptual row in the FTN table
addr	Lower end of the source address range and upper end of the source address range; the setting of source address is not supported

Output Parameters

None

Return Values

NSM_API_SET_SUCCESS when the function succeeds

NSM_API_SET_ERROR when the function fails

nsm_mpls_set_ftn_src_dst_port_max

This function sets the upper end of the source and destination port range.

Syntax

```
u_int32_t
nsm_mpls_set_ftn_src_dst_port_max (struct nsm_master *nm,
                                   u_int32_t ftn_ix, u_int32_t src_dst_port_max)
```

Input Parameters

`nm`	NSM master
`ftn_ix`	Index to a conceptual row in the FTN table
`src_dst_port_max`	Upper end of the source and destination port range; the only permitted value is 65535

Output Parameters

None

Return Values

NSM_API_SET_SUCCESS when the function succeeds

NSM_API_SET_ERROR when the function fails

nsm_mpls_set_ftn_src_dst_port_min

This function sets the lower end of the source and destination port range.

Syntax

```
u_int32_t
nsm_mpls_set_ftn_src_dst_port_min (struct nsm_master *nm,
                                   u_int32_t ftn_ix, u_int32_t src_dst_port_min)
```

Input Parameters

`nm`	NSM master
`ftn_ix`	Index to a conceptual row in the FTN table
`src_dst_port_min`	Lower end of the source and destination port range; the only permitted value is 0

Output Parameters

None

Return Values

NSM_API_SET_SUCCESS when the function succeeds

NSM_API_SET_ERROR when the function fails

© 2015 IP Infusion Inc. Proprietary

nsm_mpls_set_ftn_st_type

This function sets the storage type for this FTN.

Syntax

```
u_int32_t
nsm_mpls_set_ftn_st_type (struct nsm_master *nm,
                          u_int32_t ftn_ix, u_int32_t st_type)
```

Input Parameters

`nm`	NSM master
`ftn_ix`	Index to a conceptual row in the FTN table
`st_type`	Storage type for this FTN entry; only `ST_VOLATILE` defined in `nsm/mpls/nsm_mpls_rib.h` is permitted

Output Parameters

None

Return Values

NSM_API_SET_SUCCESS when the function succeeds

NSM_API_SET_ERROR when the function fails

nsm_mpls_set_inseg_if

This function sets the interface index for the given insegment.

Syntax

```
u_int32_t
nsm_mpls_set_inseg_if (struct nsm_master *nm, u_int32_t inseg_ix, u_int32_t inseg_if)
```

Input Parameters

`nm`	NSM master
`inseg_ix`	Insegment index
`inseg_if`	Insegment interface

Output Parameters

None

Return Values

NSM_API_SET_SUCCESS when the function succeeds

NSM_API_SET_ERROR when the function fails

nsm_mpls_set_inseg_addr_family

This function sets the address family for this insegment.

Syntax

```
u_int32_t
nsm_mpls_set_inseg_addr_family (struct nsm_master *nm,
                                u_int32_t inseg_ix, u_int32_t addr_family)
```

Input Parameters

`nm`	NSM master
`inseg_ix`	Insegment index
`addr_family`	Address family of the insegment address; the only permitted value is `INET_AD_IPV4` defined in `lib/mpls/mpls.h`

Output Parameters

None

Return Values

NSM_API_SET_SUCCESS when the function succeeds

NSM_API_SET_ERROR when the function fails

nsm_mpls_set_inseg_lb

This function sets the label for the given insegment.

Syntax

```
u_int32_t
nsm_mpls_set_inseg_lb (struct nsm_master *nm,
                       u_int32_t inseg_ix, u_int32_t in_label)
```

Input Parameters

`nm`	NSM master
`inseg_ix`	Insegment index
`in_label`	Insegment label

Output Parameters

None

Return Values

NSM_API_SET_SUCCESS when the function succeeds

NSM_API_SET_ERROR when the function fails

nsm_mpls_set_inseg_lb_ptr

This function sets the label pointer for the given insegment.

Syntax

```
u_int32_t
nsm_mpls_set_inseg_lb_ptr (struct nsm_master *nm, u_int32_t inseg_ix)
```


© 2015 IP Infusion Inc. Proprietary

Input Parameters

`nm`	NSM master
`inseg_ix`	Insegment index

Output Parameters

None

Return Values

NSM_API_SET_SUCCESS when the function succeeds

NSM_API_SET_ERROR when the function fails

nsm_mpls_set_inseg_npop

This function sets the number of labels to pop from the incoming packet.

Syntax

```
u_int32_t
nsm_mpls_set_inseg_npop (struct nsm_master *nm,
                         u_int32_t inseg_ix, u_int32_t npops)
```

Input Parameters

`nm`	NSM master
`inseg_ix`	Insegment index
`npops`	Number of labels to pop from the incoming packet; the only permitted value is 1

Output Parameters

None

Return Values

NSM_API_SET_SUCCESS when the function succeeds

NSM_API_SET_ERROR when the function fails

nsm_mpls_set_inseg_row_status

This function sets the row status of the given insegment.

Syntax

```
u_int32_t
nsm_mpls_set_inseg_row_status (struct nsm_master *nm,
                               u_int32_t inseg_ix, u_int32_t row_status)
```

Input Parameters

nm	NSM master
inseg_ix	Insegment index
row_status	Row status. These constants are from mpls_row_status enum in lib/mpls/mpls.h:

RS_ACTIVE

RS_CREATE_GO

RS_CREATE_WAIT

RS_DESTROY

RS_NOT_IN_SERVICE

RS_NOT_READY

Output Parameters

None

Return Values

NSM_API_SET_SUCCESS when the function succeeds

NSM_API_SET_ERROR when the function fails

nsm_mpls_set_inseg_st_type

This function sets the storage type of the given insegment.

Syntax

```
u_int32_t
nsm_mpls_set_inseg_st_type (struct nsm_master *nm,
                            u_int32_t inseg_ix, u_int32_t st_type)
```

Input Parameters

nm	NSM master
inseg_ix	Insegment index
st_type	Storage type; only ST_VOLATILE defined in nsm/mpls/nsm_mpls_rib.h is permitted

Output Parameters

None

© 2015 IP Infusion Inc. Proprietary

Return Values

NSM_API_SET_SUCCESS when the function succeeds

NSM_API_SET_ERROR when the function fails

nsm_mpls_set_inseg_tf_prm

This function sets the traffic parameter pointer for the given insegment.

Syntax

```
u_int32_t
nsm_mpls_set_inseg_tf_prm (struct nsm_master *nm, u_int32_t inseg_ix)
```

Input Parameters

`nm`	NSM master
`inseg_ix`	Insegment index

Output Parameters

None

Return Values

NSM_API_SET_SUCCESS when the function succeeds

NSM_API_SET_ERROR when the function fails

nsm_mpls_set_notify

This function enables or disables mplsXCUp and mplsXCDown notifications.

Syntax

```
int
nsm_mpls_set_notify (struct nsm_mpls *nmpls, int val)
```

Input Parameters

`nmpls`	The NSM MPLS master structure

`val`	Whether to enable or disable notifications; one of these constants defined in `nsm/mpls/nsm_mpls_snmp.h`:
	`NSM_MPLS_NOTIFICATIONCNTL_ENA`
	`NSM_MPLS_NOTIFICATIONCNTL_DIS`

Output Parameters

None

Return Values

NSM_API_SET_SUCCESS when the function succeeds

NSM_API_SET_ERROR when the function fails

nsm_mpls_set_outseg_if_ix

This function sets the interface index of the outsegment.

Syntax

```
u_int32_t
nsm_mpls_set_outseg_if_ix (struct nsm_master *nm,
                           u_int32_t outseg_ix, u_int32_t outseg_if_ix)
```

Input Parameters

`nm`	NSM master
`outseg_ix`	Outsegment index
`outseg_if_ix`	Interface index of the outgoing interface

Output Parameters

None

Return Values

NSM_API_SET_SUCCESS when the function succeeds

NSM_API_SET_ERROR when the function fails

nsm_mpls_set_outseg_nxt_hop_ipa

This function sets the IP address of the next hop for the given outsegment.

Syntax

```
u_int32_t
nsm_mpls_set_outseg_nxt_hop_ipa (struct nsm_master *nm, u_int32_t outseg_ix,
                                 u_int32_t length, struct pal_in4_addr addr)
```

Input Parameters

`nm`	NSM master
`outseg_ix`	Outsegment index
`length`	Length of the address
`addr`	Next hop IP address

Output Parameters

None

Return Values

NSM_API_SET_SUCCESS when the function succeeds

NSM_API_SET_ERROR when the function fails

© 2015 IP Infusion Inc. Proprietary

nsm_mpls_set_outseg_nxt_hop_ipa_type

This function sets the next hop IP address type for the given outsegment.

Syntax

```
u_int32_t
nsm_mpls_set_outseg_nxt_hop_ipa_type (struct nsm_master *nm,
                                      u_int32_t outseg_ix, u_int32_t ipa_type)
```

Input Parameters

`nm`	NSM master
`outseg_ix`	Outsegment index
`ipa_type`	Nexthop IP address type. The only value is `INET_AD_IPV4`.

Output Parameters

None

Return Values

NSM_API_SET_SUCCESS when the function succeeds

NSM_API_SET_ERROR when the function fails

nsm_mpls_set_outseg_push_top_lb

This function sets whether to push a top label onto the outgoing packets label stack. for the given outsegment.

Syntax

```
u_int32_t
nsm_mpls_set_outseg_push_top_lb (struct nsm_master *nm,
                                 u_int32_t outseg_ix, u_int32_t push_top_lb)
```

Input Parameters

`nm`	NSM master
`outseg_ix`	Outsegment index
`push_top_lb`	Whether a top label should be pushed out onto the outgoing packets label stack; the only permitted value is `NSM_TRUE` defined in `nsm/nsmd.h`

Output Parameters

None

Return Values

NSM_API_SET_SUCCESS when the function succeeds

NSM_API_SET_ERROR when the function fails

nsm_mpls_set_outseg_row_status

This function sets the row status for the given outsegment.

Syntax

```
u_int32_t
nsm_mpls_set_outseg_row_status (struct nsm_master *nm,
                                u_int32_t outseg_ix, u_int32_t row_status)
```

Input Parameters

nm	NSM master
outseg_ix	Outsegment index
row_status	Row status. These constants are from mpls_row_status enum in lib/mpls/mpls.h:

RS_ACTIVE

RS_CREATE_GO

RS_CREATE_WAIT

RS_DESTROY

RS_NOT_IN_SERVICE

RS_NOT_READY

Output Parameters

None

Return Values

NSM_API_SET_SUCCESS when the function succeeds

NSM_API_SET_ERROR when the function fails

nsm_mpls_set_outseg_st_type

This function sets the storage type for the given outsegment.

Syntax

```
u_int32_t
nsm_mpls_set_outseg_st_type (struct nsm_master *nm,
                             u_int32_t outseg_ix, u_int32_t st_type)
```

Input Parameters

nm	NSM master
outseg_ix	Outsegment index
st_type	Storage type; only ST_VOLATILE defined in nsm/mpls/nsm_mpls_rib.h is permitted

Output Parameters

None

© 2015 IP Infusion Inc. Proprietary

Return Values

NSM_API_SET_SUCCESS when the function succeeds

NSM_API_SET_ERROR when the function fails

nsm_mpls_set_outseg_tf_prm

This function sets the traffic parameters for the given outsegment.

Syntax

```
u_int32_t
nsm_mpls_set_outseg_tf_prm (struct nsm_master *nm, u_int32_t outseg_ix)
```

Input Parameters

`nm`	NSM master
`outseg_ix`	Outsegment index

Output Parameters

None

Return Values

NSM_API_SET_SUCCESS when the function succeeds

NSM_API_SET_ERROR when the function fails

nsm_mpls_set_outseg_top_lb

This function sets the outlabel to push onto the top of the outgoing packets label stack for the given outsegment.

Syntax

```
u_int32_t
nsm_mpls_set_outseg_top_lb (struct nsm_master *nm,
                            u_int32_t outseg_ix, u_int32_t out_label)
```

Input Parameters

`nm`	NSM master
`outseg_ix`	Outsegment index
`out_label`	Outlabel

Output Parameters

None

Return Values

NSM_API_SET_SUCCESS when the function succeeds

NSM_API_SET_ERROR when the function fails

nsm_mpls_set_outseg_top_lb_ptr

This function sets the label pointer for the given outsegment.

Syntax

```
u_int32_t
nsm_mpls_set_outseg_top_lb_ptr (struct nsm_master *nm, u_int32_t outseg_ix)
```

Input Parameters

`nm`	NSM master
`outseg_ix`	Outsegment index

Output Parameters

None

Return Values

NSM_API_SET_SUCCESS when the function succeeds

NSM_API_SET_ERROR when the function fails

nsm_mpls_set_xc_adm_status

This function sets the operational status of the given cross connect.

Syntax

```
u_int32_t
nsm_mpls_set_xc_adm_status (struct nsm_master *nm,
                            u_int32_t xc_ix, u_int32_t inseg_ix,
                            u_int32_t outseg_ix, u_int32_t admn_status)
```

Input Parameters

`nm`	NSM master
`outseg_ix`	Outsegment index
`inseg_ix`	Insegment index
`xc_ix`	Cross connect index

`admn_status`	Operational status. These constants are from the `mpls_admn_status` enum in `lib/mpls/mpls.h`:
	`ADMN_UP`
	`ADMN_DOWN`
	`ADMN_TESTING`

Output Parameters

None

Return Values

NSM_API_SET_SUCCESS when the function succeeds

© 2015 IP Infusion Inc. Proprietary

NSM_API_SET_ERROR when the function fails

nsm_mpls_set_xc_lb_stk_ix

This function sets the primary index that identifies a stack of labels to push below the top label.

Syntax

```
u_int32_t
nsm_mpls_set_xc_lb_stk_ix (struct nsm_master *nm,
                           u_int32_t xc_ix, u_int32_t inseg_ix,
                           u_int32_t outseg_ix, u_int32_t lb_stk_ix)
```

Input Parameters

nm	NSM master
outseg_ix	Outsegment index
inseg_ix	Insegment index
xc_ix	Cross connect index
lb_stk_ix	Primary index

Output Parameters

None

Return Values

NSM_API_SET_SUCCESS when the function succeeds

NSM_API_SET_ERROR when the function fails

nsm_mpls_set_xc_lspid

This function sets the label switched path for the given cross connect.

Syntax

```
u_int32_t
nsm_mpls_set_xc_lspid (struct nsm_master *nm,
                       u_int32_t xc_ix, u_int32_t inseg_ix,
                       u_int32_t outseg_ix)
```

Input Parameters

nm	NSM master
outseg_ix	Outsegment index
inseg_ix	Insegment index
xc_ix	Cross connect index

Output Parameters

None

Return Values

NSM_API_SET_SUCCESS when the function succeeds

NSM_API_SET_ERROR when the function fails

nsm_mpls_set_xc_row_status

This function sets the row status of the given cross connect.

Syntax

```
u_int32_t
nsm_mpls_set_xc_row_status (struct nsm_master *nm,
                            u_int32_t xc_ix, u_int32_t inseg_ix,
                            u_int32_t outseg_ix, u_int32_t row_status)
```

Input Parameters

`nm`	NSM master
`outseg_ix`	Outsegment index
`inseg_ix`	Insegment index
`xc_ix`	Cross connect index
`row_status`	Row status. Constants are from the `mpls_row_status` enum in `lib/mpls/mpls.h`:
	`RS_ACTIVE`
	`RS_CREATE_GO`
	`RS_CREATE_WAIT`
	`RS_DESTROY`
	`RS_NOT_IN_SERVICE`
	`RS_NOT_READY`

Output Parameters

None

Return Values

NSM_API_SET_SUCCESS when the function succeeds

NSM_API_SET_ERROR when the function fails

nsm_mpls_set_xc_st_type

This function sets the storage type of the cross connect entry.

Syntax

```
u_int32_t
nsm_mpls_set_xc_st_type (struct nsm_master *nm,
                         u_int32_t xc_ix, u_int32_t inseg_ix,
                         u_int32_t outseg_ix, u_int32_t st_type)
```


© 2015 IP Infusion Inc. Proprietary

Input Parameters

`nm`	NSM master
`outseg_ix`	Outsegment index
`inseg_ix`	Insegment index
`xc_ix`	Cross connect index
`st_type`	Storage type, only `ST_VOLATILE` defined in `nsm/mpls/nsm_mpls_rib.h` is permitted

Output Parameters

None

Return Values

NSM_API_SET_SUCCESS when the function succeeds

NSM_API_SET_ERROR when the function fails

© 2015 IP Infusion Inc. Proprietary

CHAPTER 16 MPLS Command API

This chapter describes the messages and functions that support Multi-Protocol Label Switching (MPLS). MPLS is implemented in NSM according to RFC 3031: Multi-Protocol Label Switching Architecture.

MPLS Messages

The following messages define communications between the NSM and MPLS.

FTN Messages

FTN messages are defined in the sections that follow.

FTN Add Request

Protocols send this message to NSM to install FTN entries. Every message contains a message ID, a protocol ID, and a protocol-specific lookup key. These elements are returned by NSM in the corresponding FTN Add Reply or notification messages.

FTN Add Reply

NSM sends this message to the protocols as a confirmation for the corresponding FTN Add Request message. This message contains the same message ID as that of the corresponding FTN Add Request message.

FTN Delete Request

Protocols send this message to the NSM to remove FTN entries. Every message contains a message ID, protocol ID, and a protocol-specific lookup key. The ZebOS-XP MIB defines two message types for deleting FTN entries: FTN Delete Regular and FTN Delete Slow.

FTN Delete Regular

This message deletes FTN entries for which a confirmation FTN Add Reply message has been received by the protocols. The message contains the VRF ID, FEC prefix, and FTN index.

FTN Delete Slow

This message deletes FTN entries for which no confirmation FTN Add Reply message has been received by the protocol. All data elements sent during the corresponding FTN Add Request message are sent along with this message.

ILM Messages

ILM messages are defined in the sections that follow.

ILM Add Request

Protocols send this message to NSM to install ILM entries. Every message contains a message ID, protocol ID, and a protocol-specific lookup key. The NSM returns these elements in the corresponding ILM Add Reply or notification messages.

ILM Add Reply

NSM sends this message to the protocols to serve as confirmation for the corresponding ILM Add Request message. This message contains the same message ID as that of the corresponding ILM Add Request message.

ILM Delete Request

This message deletes ILM entries. It contains an Incoming label, an incoming interface along with a message ID, a protocol ID, and a lookup key used in the corresponding ILM Add Request message.

Virtual Circuit Messages

Virtual Circuit messages are defined in the sections that follow.

VC FTN Add Request

This message adds an FTN entry for an MPLS-based Layer-2 Virtual Circuit (VC). Every message contains a message ID, protocol ID, and a protocol-specific lookup key. NSM returns these parameters' elements in the corresponding VC FTN Add Reply or notification messages.

VC ILM Add Request

Protocols send this message to the NSM to install ILM entries for MPLS Layer-2 VCs. Every message contains a message ID, protocol ID, and a protocol-specific lookup key. NSM returns these parameters in the corresponding VC ILM Add Reply or notification messages.

VC FTN Delete Request

Protocols send this message to the NSM to remove VC FTN entries. Every message contains a message ID, protocol ID, and a protocol-specific lookup key.

VC ILM Delete Request

This message deletes VC ILM entries. It contains the incoming label and interface, along with the message ID, protocol ID, and lookup key used in the corresponding VC ILM Add Request message.

VC FTN Add Reply

NSM sends this message to protocols. It serves as confirmation for the corresponding VC FTN Add Request message. This message must contain the same message ID as that of the corresponding VC FTN Add Request message.

VC ILM Add Reply

NSM sends this message to protocols to serve as confirmation for the corresponding VC ILM Add Request message. This message must contain the same message ID as that of the corresponding VC ILM Add Request message.

Notification Messages

The NSM and signaling modules exchange these messages to indicate errors and other exceptional conditions. These messages are generated under the following conditions.

FTN Add Failure

NSM sends this message to notify the protocols of a failure when adding FTN entries.

© 2015 IP Infusion Inc. Proprietary

ILM Add Failure

NSM sends this message to notify the protocols of a failure when adding ILM entries.

VC FTN Add Failure

The NSM sends this message to notify the protocols of a failure in adding VC FTN entries.

VC ILM Add Failure

The NSM sends this message to notify the protocols of a failure in adding VC ILM entries.

Command API

The table contains the functions for Multi-Protocol Label Switching commands.

Function	Description
nsm_gmpls_api_add_lsp_tunnel	Adds an LSP tunnel
nsm_gmpls_api_add_mapped_route	Adds a mapped route
nsm_gmpls_api_del_mapped_route	Deletes a mapped route
nsm_mpls_api_admin_group_add	Adds an administrative group
nsm_mpls_api_admin_group_del	Deletes an administrative group
nsm_mpls_api_del_lsp_tunnel	Deletes an administrative group
nsm_mpls_api_del_lsp_tunnel	Deletes an LSP tunnel
nsm_mpls_api_disable_all_interfaces	Disables all interfaces for MPLS capabilities
nsm_mpls_api_egress_ttl_set	Sets a custom TTL for LSPs for which this LSR is the egress
nsm_mpls_api_enable_all_interfaces	Enables all interfaces for MPLS capabilities
nsm_mpls_api_ingress_ttl_set	Sets a custom TTL for LSPs for which this LSR is the egress
nsm_mpls_api_local_pkt_handling	Enables or disables local packet handling of locally generated TCP packets
nsm_mpls_api_ls_max_label_val_set	Sets the maximum label value for label space
nsm_mpls_api_ls_max_label_val_unset	Resets the maximum label value for label space
nsm_mpls_api_ls_min_label_val_set	Sets the minimum label value for label space
nsm_mpls_api_ls_min_label_val_unset	Resets the minimum label value for label space
nsm_mpls_api_max_label_val_set	Sets the maximum label value for all interfaces
nsm_mpls_api_max_label_val_unset	Resets the maximum label value for all interfaces
nsm_mpls_api_min_label_val_set	Sets the minimum label value for all interfaces
nsm_mpls_api_min_label_val_unset	Resets the minimum label value for all interfaces

© 2015 IP Infusion Inc. Proprietary

Function	Description
nsm_mpls_api_pipe_model_update	Sets the LSP model to pipe or uniform
nsm_mpls_api_ttl_propagate_cap_update	Enables or disables TTL propagation
nsm_mpls_check_valid_interface	Checks if the interface is valid for MPLS

Include File

To call the functions in the API, you must include `nsm/mpls/nsm_mpls_api.h`.

nsm_gmpls_api_add_lsp_tunnel

This function adds an LSP tunnel.

Syntax

```
s_int32_t
nsm_gmpls_api_add_lsp_tunnel (struct nsm_master *nm, struct interface *ifp,
                              u_int32_t il,
                              u_int32_t ol,
                              struct prefix *pfx)
```

Input Parameters

`nm`	A pointer to NSM master structure
`ifp`	Pointer to the interface
`il`	Incoming Label
`ol`	Outgoing Label
`pfx`	Prefix

Output Parameters

None

Return Values

NSM_SUCCESS when the LSP tunnel is added successfully

NSM_ERR_MEM_ALLOC_FAILURE when memory allocation fails

NSM_ERR_EXISTS when the LSP tunnel already exists

nsm_gmpls_api_add_mapped_route

This function adds a mapped route.

Syntax

```
s_int32_t
nsm_gmpls_api_add_mapped_route (struct nsm_master *nm,
                                struct fec_gen_entry *ent, struct prefix *fec)
```


© 2015 IP Infusion Inc. Proprietary

Input Parameters

`nm`	A pointer to NSM master structure
`ent`	FEC generic entry
`fec`	Forward Equivalence Class

Output Parameters

None

Return Values

NSM_FAILURE when the node is not present in the mapped route table

NSM_ERR_MEM_ALLOC_FAILURE when memory allocation fails

NSM_SUCCESS when the mapped route is added successfully

nsm_gmpls_api_del_mapped_route

This function deletes a mapped route.

Syntax

```
s_int32_t
nsm_gmpls_api_del_mapped_route (struct nsm_master *nm,
                                struct fec_gen_entry *ent, struct prefix *fec)
```

Input Parameters

`nm`	A pointer to NSM master structure
`ent`	FEC generic entry
`fec`	Forward Equivalence Class

Output Parameters

None

Return Values

NSM_FAILURE when the node is not present in the mapped route table

NSM_ERR_INVALID_ARGS when the FEC is invalid

NSM_SUCCESS when the mapped route is deleted successfully

nsm_mpls_api_admin_group_add

This function adds an administrative group.

Syntax

```
int
nsm_mpls_api_admin_group_add (struct nsm_master *nm, char *name, int val)
```

Input Parameters

`nm`	A pointer to NSM master structure

`name`	Administrative group name
`val`	Administrative group value

Output Parameters

None

Return Values

NSM_FAILURE when the function fails

NSM_ERR_NAME_TOO_LONG when the given name is greater than the maximum size

ADMIN_GROUP_ERR_DUPLICATE when the given name already exists or is already in use

ADMIN_GROUP_ERR_EXIST when some other name already exists

NSM_SUCCESS when the mapped route is added successfully

nsm_mpls_api_admin_group_del

This function deletes an administrative group.

Syntax

```
int
nsm_mpls_api_admin_group_del (struct nsm_master *nm, char *name, int val)
```

Input Parameters

`nm`	A pointer to NSM master structure
`name`	Administrative group name
`val`	Administrative group value

Output Parameters

None

Return Values

NSM_FAILURE when the function fails

NSM_ERR_NAME_TOO_LONG when the given name is greater than the maximum size

ADMIN_GROUP_ERR_EEXIST when the given name already exists

NSM_ERR_NAME_IN_USE when the administrative group is in use in the interface configuration

NSM_SUCCESS when the group is deleted successfully

nsm_mpls_api_del_lsp_tunnel

This function deletes an LSP tunnel.

Syntax

```
int
nsm_mpls_api_del_lsp_tunnel (struct nsm_master *nm,
                             struct interface *ifp, u_int32_t in_label)
```

 © 2015 IP Infusion Inc. Proprietary

Input Parameters

`nm`	A pointer to NSM master structure
`ifp`	Pointer to the interface
`in_label`	Incoming label

Output Parameters

None

Return Values

NSM_ERR_NOT_FOUND when the LSP tunnel is not found

NSM_SUCCESS when the LSP tunnel is deleted successfully

nsm_mpls_api_disable_all_interfaces

This function disables all interfaces for MPLS capabilities.

Syntax

```
void
nsm_mpls_api_disable_all_interfaces (struct nsm_master *nm)
```

Input Parameters

`nm`	A pointer to NSM master structure

Output Parameters

None

Return Values

None

nsm_mpls_api_egress_ttl_set

This function sets a custom TTL for LSPs for which this LSR is the egress.

Syntax

```
void
nsm_mpls_api_egress_ttl_set (struct nsm_master *nm, int ttl_val)
```

Input Parameters

`nm`	A pointer to NSM master structure
`ttl_val`	Ingress TTL

Output Parameters

None

Return Values

None

nsm_mpls_api_enable_all_interfaces

This function enables all interfaces for MPLS capabilities.

Syntax

```
void
nsm_mpls_api_enable_all_interfaces (struct nsm_master *nm,
                                    u_int16_t label_space)
```

Input Parameters

nm	A pointer to NSM master structure
label_space	Label space

Output Parameters

None

nsm_mpls_api_ingress_ttl_set

This function sets the custom TTL for LSPs for which this LSR is the ingress.

Syntax

```
void
nsm_mpls_api_ingress_ttl_set (struct nsm_master *nm, int ttl_val)
```

Input Parameters

nm	A pointer to NSM master structure
ttl_val	Ingress TTL

Output Parameters

None

Return Values

None

nsm_mpls_api_local_pkt_handling

This function enables or disables local packet handling of locally generated TCP packets.

Syntax

```
void
nsm_mpls_api_local_pkt_handling (struct nsm_master *nm, bool_t bool)
```

Input Parameters

nm	A pointer to NSM master structure
bool	Whether to enable local packet handling; one of these constants from nsm/nsmd.h:
NSM_TRUE	Enable local packet handling
NSM_FALSE	Disable local packet handling

© 2015 IP Infusion Inc. Proprietary

Output Parameters

None

Return Values

None

nsm_mpls_api_ls_max_label_val_set

This function sets the maximum label value for label space.

Syntax

```
int
nsm_mpls_api_ls_max_label_val_set (struct nsm_master *nm,
                                   u_int16_t label_space, u_int32_t val)
```

Input Parameters

`nm`	A pointer to NSM master structure
`label_space`	Label space
`val`	Max label value to set

Output Parameters

None

Return Values

NSM_SUCCESS when the function succeeds

NSM_ERR_LS_IN_USE when any label space is in use

nsm_mpls_api_ls_max_label_val_unset

This function resets the maximum label value for label space.

Syntax

```
int
nsm_mpls_api_ls_max_label_val_unset (struct nsm_master *nm,
                                     u_int16_t label_space)
```

Input Parameters

`nm`	A pointer to NSM master structure
`label_space`	Label space

Output Parameters

None

Return Values

NSM_SUCCESS when the function succeeds

NSM_ERR_LS_IN_USE when any label space is in use

nsm_mpls_api_ls_min_label_val_set

This function sets the minimum label value for label space.

Syntax

```
int
nsm_mpls_api_ls_min_label_val_set (struct nsm_master *nm,
                                   u_int16_t label_space, u_int32_t val)
```

Input Parameters

`nm`	A pointer to NSM master structure
`label_space`	Label space
`val`	Minimum label value

Output Parameters

None

Return Values

NSM_SUCCESS when the function succeeds

NSM_ERR_LS_IN_USE when any label space is in use

nsm_mpls_api_ls_min_label_val_unset

This function resets the minimum label value for label space.

Syntax

```
int
nsm_mpls_api_ls_min_label_val_unset (struct nsm_master *nm,
                                     u_int16_t label_space)
```

Input Parameters

`nm`	A pointer to NSM master structure
`label_space`	Label space

Output Parameters

None

Return Values

NSM_SUCCESS when the function succeeds

NSM_ERR_LS_IN_USE when any label space is in use

nsm_mpls_api_max_label_val_set

This function sets the maximum label value for all interfaces.

© 2015 IP Infusion Inc. Proprietary

Syntax

```
int
nsm_mpls_api_max_label_val_set (struct nsm_master *nm, u_int32_t val)
```

Input Parameters

`nm`	A pointer to NSM master structure
`val`	Maximum label value

Output Parameters

None

Return Values

NSM_SUCCESS when the function succeeds

NSM_ERR_LS_INVALID_MAX_LABEL when the label value is less than the minimum allowable

NSM_ERR_LS_IN_USE when any label space is in use

nsm_mpls_api_max_label_val_unset

This function resets the maximum label value for all interfaces.

Syntax

```
int
nsm_mpls_api_max_label_val_unset (struct nsm_master *nm)
```

Input Parameters

`nm`	A pointer to NSM master structure

Output Parameters

None

Return Values

NSM_SUCCESS when the function succeeds

NSM_ERR_LS_IN_USE when any label space is in use

nsm_mpls_api_min_label_val_set

This function sets the minimum label value for all interfaces.

Syntax

```
int
nsm_mpls_api_min_label_val_set (struct nsm_master *nm, u_int32_t val)
```

Input Parameters

`nm`	A pointer to NSM master structure
`val`	Minimum label value

Output Parameters

None

Return Values

NSM_SUCCESS when the function succeeds

NSM_ERR_LS_IN_USE when any label space is in use

nsm_mpls_api_min_label_val_unset

This function resets the minimum label value for all interfaces.

Syntax

```
int
nsm_mpls_api_min_label_val_unset (struct nsm_master *nm)
```

Input Parameters

`nm`	A pointer to NSM master structure

Output Parameters

None

Return Values

NSM_SUCCESS when the function succeeds

NSM_ERR_LS_IN_USE when any label space is in use

nsm_mpls_api_pipe_model_update

This function sets the LSP model to pipe or uniform.

Syntax

```
void
nsm_mpls_api_pipe_model_update (struct nsm_master *nm, bool_t set)
```

Input Parameters

`nm`	A pointer to NSM master structure
`set`	Pipe or uniform model; one of these constants from `nsm/nsmd.h`:
`NSM_TRUE`	Pipe model
`NSM_FALSE`	Uniform model

Output Parameters

None

Return Values

None

© 2015 IP Infusion Inc. Proprietary

nsm_mpls_api_ttl_propagate_cap_update

This function enables or disables TTL propagation.

Syntax

```
void
nsm_mpls_api_ttl_propagate_cap_update (struct nsm_master *nm, bool_t set)
```

Input Parameters

`nm`	A pointer to NSM master structure.
`set`	Whether to enable TTL propagation; one of these constants from `nsm/nsmd.h`:
`NSM_TRUE`	Enable TTL propagation
`NSM_FALSE`	Disable TTL propagation

Output Parameters

None

Return Values

None

nsm_mpls_check_valid_interface

This function checks whether the interface is valid for MPLS.

Syntax

```
int
nsm_mpls_check_valid_interface (struct interface *ifp, u_char opcode)
```

Input Parameters

`ifp`	Pointer to the interface
`opcode`	Label operational code; one of these constants from `lib/mpls_client/mpls_common.h`:
	PUSH
	`POP`
	`SWAP`
	SWAP
	POP_FOR_VPN
	DLVR_TO_IP
	PUSH_AND_LOOKUP
	PUSH_FOR_VC
	PUSH_AND_LOOKUP_FOR_VC
	POP_FOR_VC
	SWAP_AND_LOOKUP
	MPLS_NO_OP

FTN_LOOKUP

PUSH_FOR_TP_VC

Output Parameters

None

Return Values

NSM_TRUE when the interface is valid for MPLS

NSM_FALSE when the interface is not valid for MPLS

© 2015 IP Infusion Inc. Proprietary

CHAPTER 17 MPLS Pseudowire MIB API

This chapter describes the Management Information Base (MIB) support for MPLS pseudowires in NSM. It contains the functions for Pseudo-Wire MPLS MIB table entries in ZebOS-XP.

Overview

The PseudoWire (PW) MIB is implemented as an MPLS feature in ZebOS-XP, based on these standards:

- RFC 5601 PW MIB
- RFC 5602 PW over MPLS PSN MIB
- RFC 5603 Ethernet PW MIB

The tables contain cross-reference links to the Functions that are used for each Object Type.

RFC 5601

RFC 5601 defines objects for modeling PW edge-to-edge services carried over a *general* packet-switched network.

Object Type	Syntax	Access	Functions
pwType	IANAPwTypeTC	read-create	nsm_mpls_set_pw_type
pwOwner	INTEGER	read-create	nsm_mpls_set_pw_owner
pwPsnType	IANAPwPsnTypeTC	read-create	nsm_mpls_set_pw_psn_type
pwSetUpPriority	Integer32	read-create	nsm_mpls_set_pw_setup_prty
pwHoldingPriority	Integer32	read-create	nsm_mpls_set_pw_hold_prty
pwPeerAddr	InetAddress	read-create	nsm_mpls_set_pw_peer_addr
pwIfIndex	InterfaceIndexOrZero	read-create	nsm_mpls_set_pw_if_ix
pwID	PwIDType	read-create	nsm_mpls_set_pw_id
pwLocalGroupID	PwGroupID	read-create	nsm_mpls_set_pw_local_grp_id
pwGroupAttachmentID	PwAttachmentIdentifierType	read-create	nsm_mpls_set_pw_grp_attchmt_id
pwLocalAttachmentID	PwAttachmentIdentifierType	read-create	nsm_mpls_set_pw_local_attchmt_id
eAttachmentID	PwAttachmentIdentifierType	read-create	nsm_mpls_set_pw_peer_attchmt_id
pwCwPreference	TruthValue	read-create	nsm_mpls_set_pw_cw_prfrnce
pwLocalIfMtu	Unsigned32	read-create	nsm_mpls_set_pw_local_if_mtu
pwLocalIfString	TruthValue	read-create	nsm_mpls_set_pw_local_if_string

Object Type	Syntax	Access	Functions
pwLocalCapabAdvert	IANAPwCapabilities	read-create	nsm_mpls_set_pw_local_capab_advrt
pwFragmentCfgSize	PwFragSize	read-create	nsm_mpls_set_pw_frgmt_cfg_size
pwFcsRetentionCfg	INTEGER	read-create	nsm_mpls_set_pw_fcs_retentn_cfg
pwOutboundLabel	Unsigned32	read-create	nsm_mpls_set_pw_outbd_label
pwInboundLabel	Unsigned32	read-create	nsm_mpls_set_pw_inbd_label
pwName	SnmpAdminString	read-create	nsm_mpls_set_pw_name
pwDescr	SnmpAdminString	read-create	nsm_mpls_set_pw_descr
pwAdminStatus	INTEGER	read-create	nsm_mpls_set_pw_admin_status
pwRowStatus	RowStatus	read-create	nsm_mpls_set_pw_row_status
pwStorageType	StorageType	read-create	nsm_mpls_set_pw_st_type
pwOamEnable	TruthValue	read-create	nsm_mpls_set_pw_oam_enable
pwGenAGIType	PwGenIdType	read-create	nsm_mpls_set_pw_gen_agi_type
pwGenLocalAIIType	PwGenIdType	read-create	nsm_mpls_set_pw_gen_loc_aii_type
pwGenRemoteAIIType	PwGenIdType	read-create	nsm_mpls_set_pw_gen_rem_aii_type
pwUpDownNotifEnable	TruthValue	read-write	nsm_mpls_get_pw_up_dn_notify nsm_mpls_set_pw_up_dn_notify
pwDeletedNotifEnable	TruthValue	read-write	nsm_mpls_get_pw_del_notify nsm_mpls_set_pw_del_notify
pwNotifRate	Unsigned32	read-write	nsm_mpls_set_pw_notify_rate

RFC 5602

RFC 5602 defines objects for modeling PW edge-to-edge services carried over an *MPLS* network.

Object Type	Syntax	Access	Function
pwMplsMplsType	BITS	read-write	nsm_mpls_set_pw_mpls_mpls_type
pwMplsExpBitsMode	INTEGER	read-write	nsm_mpls_set_pw_mpls_exp_bits_mode
pwMplsExpBits	Unsigned32	read-write	nsm_mpls_set_pw_mpls_exp_bits
pwMplsTtl	Unsigned32	read-write	nsm_mpls_set_pw_mpls_ttl
pwMplsLocalLdpID	MplsLdpIdentifier	read-write	nsm_mpls_set_pw_mpls_lcl_ldp_id
pwMplsLocalLdpEntityIndex	Unsigned32	read-write	nsm_mpls_set_pw_mpls_lcl_ldp_entty_ix
pwMplsOutboundLsrXcIndex	MplsIndexType	read-write	nsm_mpls_set_pw_mpls_outbd_lsr_xc_ix

© 2015 IP Infusion Inc. Proprietary

Object Type	Syntax	Access	Function
pwMplsOutboundTunnelIndex	MplsTunnelIndex	read-write	nsm_mpls_set_pw_mpls_outbd_tnl_ix
pwMplsOutboundTunnelLclLSR	MplsLsrIdentifier	read-write	nsm_mpls_set_pw_mpls_outbd_tnl_lcl_lsr
pwMplsOutboundTunnelPeerLSR	MplsLsrIdentifier	read-write	nsm_mpls_set_pw_mpls_outbd_tnl_peer_lsr
pwMplsOutboundIfIndex	InterfaceIndexOrZero	read-write	nsm_mpls_set_pw_mpls_outbd_tnl_ix

RFC 5603

RFC 5063 defines objects for modelling Ethernet PW services. The PW Ethernet (ENET) table provides Ethernet port mapping and VLAN configuration for each Ethernet PW.

PW Ethernet Table

Object Type	Syntax	Access	Functions
pwEnetPwVlan	VlanIdOrAnyOrNone	read-create	nsm_mpls_set_pw_enet_pw_vlan
pwEnetVlanMode	INTEGER	read-create	nsm_mpls_set_pw_enet_vlan_mode
pwEnetPortVlan	VlanIdOrAnyOrNone	read-create	nsm_mpls_set_pw_enet_port_vlan
pwEnetPortIfIndex	InterfaceIndexOrZero	read-create	nsm_mpls_set_pw_enet_port_if_index
pwEnetPwIfIndex	InterfaceIndexOrZero	read-create	nsm_mpls_set_pw_enet_pw_if_index
pwEnetRowStatus	RowStatus	read-create	nsm_mpls_set_pw_enet_row_status
pwEnetStorageType	StorageType	read-create	nsm_mpls_set_pw_enet_storage_type

MIB API

This section contains the functions in the MPLS pseudowire MIB API.

Function	Description
nsm_mpls_get_pw_del_notify	Gets the delete notification setting
nsm_mpls_get_pw_up_dn_notify	Gets the up down notification setting
nsm_mpls_set_pw_admin_status	Sets the administrative status for a virtual circuit
nsm_mpls_set_pw_attchd_pw_ix	Sets the attached pseudowire index for a virtual circuit
nsm_mpls_set_pw_cw_prfrnce	Sets the control word preference for a virtual circuit
nsm_mpls_set_pw_del_notify	Enables or disables delete notification
nsm_mpls_set_pw_descr	Sets the description for a virtual circuit

© 2015 IP Infusion Inc. Proprietary

Function	Description
nsm_mpls_set_pw_enet_port_if_index	Sets the interface index
nsm_mpls_set_pw_enet_port_vlan	Sets the identifier of the port VLAN
nsm_mpls_set_pw_enet_pw_if_index	Sets the interface index
nsm_mpls_set_pw_enet_pw_vlan	Sets the identifier of the ENET VLAN
nsm_mpls_set_pw_enet_row_status	Sets the row the status
nsm_mpls_set_pw_enet_storage_type	Sets the storage type
nsm_mpls_set_pw_enet_vlan_mode	Sets the VLAN mode value
nsm_mpls_set_pw_fcs_retentn_cfg	Sets the FCS retention configuration for a virtual circuit
nsm_mpls_set_pw_frgmt_cfg_size	Sets the type for a virtual circuit
nsm_mpls_set_pw_gen_agi_type	Sets the attachment group identifier (AGI) type for a virtual circuit
nsm_mpls_set_pw_gen_loc_aii_type	Sets the local forwarder attachment individual identifier (AII) type for a virtual circuit
nsm_mpls_set_pw_gen_rem_aii_type	Sets the remote forwarder attachment individual identifier (AII) type for a virtual circuit
nsm_mpls_set_pw_grp_attchmt_id	Sets the group attachment identifier for a virtual circuit
nsm_mpls_set_pw_hold_prty	Sets the hold priority for a virtual circuit
nsm_mpls_set_pw_id	Sets the identifier for a virtual circuit
nsm_mpls_set_pw_if_ix	Sets the interface index for a virtual circuit
nsm_mpls_set_pw_inbd_label	Sets the inbound label for a virtual circuit
nsm_mpls_set_pw_local_attchmt_id	Sets the local attachment ID for a virtual circuit
nsm_mpls_set_pw_local_capab_advrt	Sets the local capabilities to be advertised for a virtual circuit
nsm_mpls_set_pw_local_grp_id	Sets the local group identifier for a virtual circuit
nsm_mpls_set_pw_local_if_string	Sets the local interface string for a virtual circuit
nsm_mpls_set_pw_mpls_exp_bits	Sets the type of shim label EXP bit for a virtual circuit
nsm_mpls_set_pw_mpls_exp_bits_mode	Sets the shim label's EXP bit mode for a virtual circuit
nsm_mpls_set_pw_mpls_lcl_ldp_entty_ix	Sets the local LDP entity index for a virtual circuit
nsm_mpls_set_pw_mpls_lcl_ldp_id	Sets the local identifier of the LDP entity which creates the PW for a virtual circuit
nsm_mpls_set_pw_mpls_mpls_type	Sets the outer tunnel type for a virtual circuit
nsm_mpls_set_pw_mpls_outbd_if_ix	Sets the outbound interface index for a virtual circuit
nsm_mpls_set_pw_mpls_outbd_lsr_xc_ix	Sets the outbound tunnel cross connect index for a virtual circuit

© 2015 IP Infusion Inc. Proprietary

Function	Description
nsm_mpls_set_pw_mpls_outbd_tnl_ix	Sets the outbound tunnel index for a virtual circuit
nsm_mpls_set_pw_mpls_outbd_tnl_lcl_lsr	Sets the outbound tunnel LSR identifier for a virtual circuit
nsm_mpls_set_pw_mpls_outbd_tnl_peer_lsr	Sets the peer outbound tunnel LSR address for a virtual circuit
nsm_mpls_set_pw_mpls_ttl	Sets the TTL on a PW shim for a virtual circuit
nsm_mpls_set_pw_name	Sets the name for a virtual circuit
nsm_mpls_set_pw_notify_rate	Sets the notification rate
nsm_mpls_set_pw_oam_enable	Enables or disables PW OAM for a virtual circuit
nsm_mpls_set_pw_outbd_label	Sets the outbound label for a virtual circuit
nsm_mpls_set_pw_owner	Sets the owner for a virtual circuit
nsm_mpls_set_pw_peer_addr	Sets the peer address for a virtual circuit
nsm_mpls_set_pw_peer_attchmt_id	Sets the peer attachment ID for a virtual circuit
nsm_mpls_set_pw_psn_type	Sets the PSN type for a virtual circuit
nsm_mpls_set_pw_row_status	Sets the row status for a virtual circuit
nsm_mpls_set_pw_setup_prty	Sets the setup priority for a virtual circuit
nsm_mpls_set_pw_st_type	Sets the storage type for a virtual circuit
nsm_mpls_set_pw_type	Sets the type for a virtual circuit
nsm_mpls_set_pw_up_dn_notify	Enables or disables up down notification

Include Files

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

- `nsm/mpls/nsm_mpls_vc_api.h`
- `nsm/mpls/nsm_mpls_vc_snmp.h`

nsm_mpls_get_pw_del_notify

This function gets the delete notification setting.

Syntax

```
#include "nsm/mpls/nsm_mpls_vc_api.h"
u_int32_t
nsm_mpls_get_pw_del_notify (struct nsm_master *nm, int *val);
```

Input Parameters

`nm` Pointer to the NSM master structure

Output Parameters

`val`	Pointer to the notification setting

Return Value

NSM_API_GET_ERROR when the pointer to the NSM master structure is NULL

NSM_API_GET_SUCCESS when the function succeeds

nsm_mpls_get_pw_up_dn_notify

This function gets the up down notification setting.

Syntax

```
#include "nsm/mpls/nsm_mpls_vc_api.h"
u_int32_t
nsm_mpls_get_pw_up_dn_notify (struct nsm_master *nm, int *val);
```

Input Parameters

`nm`	Pointer to the NSM master structure

Output Parameters

`val`	Pointer to the notification setting

Return Value

NSM_API_GET_ERROR when the pointer to NSM master structure is NULL

NSM_API_GET_SUCCESS when the function succeeds

nsm_mpls_set_pw_admin_status

This function sets the administrative status for a virtual circuit.

Syntax

```
#include "nsm/mpls/nsm_mpls_vc_snmp.h"
u_int32_t
nsm_mpls_set_pw_admin_status (struct nsm_master *nm,
                              u_int32_t pw_ix, u_int32_t flag);
```

Input Parameters

`nm`	Pointer to the NSM master structure
`pw_ix`	Index to the virtual circuit table
`flag`	Status

Output Parameters

None

Return Value

NSM_API_SET_ERROR when the PW index is not valid

© 2015 IP Infusion Inc. Proprietary

NSM_API_SET_SUCCESS when the function succeeds

nsm_mpls_set_pw_attchd_pw_ix

This function sets the attached pseudowire index for a virtual circuit.

Syntax

```
#include "nsm/mpls/nsm_mpls_vc_snmp.h"
u_int32_t
nsm_mpls_set_pw_attchd_pw_ix (struct nsm_master *nm,
                              u_int32_t pw_ix, u_int32_t attchd_ix);
```

Input Parameters

nm	Pointer to the NSM master structure
pw_ix	Index to the virtual circuit table
attchd_ix	Index

Output Parameters

None

Return Value

NSM_API_SET_ERROR when the PW index is not valid

NSM_API_SET_SUCCESS when the function succeeds

nsm_mpls_set_pw_cw_prfrnce

This function sets the control word preference for a virtual circuit.

Syntax

```
#include "nsm/mpls/nsm_mpls_vc_snmp.h"
u_int32_t
nsm_mpls_set_pw_cw_prfrnce (struct nsm_master *nm,
                            u_int32_t pw_ix, u_int32_t cw);
```

Input Parameters

nm	Pointer to the NSM master structure
pw_ix	Index to the virtual circuit table
cw	Control word preference

Output Parameters

None

Return Value

NSM_API_SET_ERROR when the PW index is not valid

NSM_API_SET_SUCCESS when the function succeeds

nsm_mpls_set_pw_del_notify

This function enables or disables delete notification.

Syntax

```
#include "nsm/mpls/nsm_mpls_vc_api.h"
u_int32_t
nsm_mpls_set_pw_del_notify (struct nsm_mpls *nmpls, int val);
```

Input Parameters

`nmpls`	NSM MPLS pointer
`val`	PAL_TRUE or PAL_FALSE

Output Parameters

None

Return Value

NSM_API_SET_ERROR when the NSM MPLS pointer is NULL

NSM_API_GET_SUCCESS when the function succeeds

nsm_mpls_set_pw_descr

This function sets the description for a virtual circuit.

Syntax

```
#include "nsm/mpls/nsm_mpls_vc_snmp.h"
u_int32_t
nsm_mpls_set_pw_descr (struct nsm_master *nm,
                       u_int32_t pw_ix, char *name);
```

Input Parameters

`nm`	Pointer to the NSM master structure
`pw_ix`	Index to the virtual circuit table
`name`	Description

Output Parameters

None

Return Value

NSM_API_SET_ERROR when the PW index is not valid

NSM_API_SET_SUCCESS when the function succeeds

nsm_mpls_set_pw_enet_port_if_index

This function sets the interface index.

© 2015 IP Infusion Inc. Proprietary

Syntax

```
#include "nsm/mpls/nsm_mpls_vc_snmp.h"
u_int32_t
nsm_mpls_set_pw_enet_port_if_index (struct nsm_master *nm,
                                    u_int32_t pw_ix, u_int32_t pw_instance,
                                    u_int32_t ifindex)
```

Input Parameters

`nm`	Pointer to the NSM master structure
`pw_ix`	Index to the virtual circuit table
`pw_instance`	Index to the row in the PW ENET table
`ifindex`	Interface index

Output Parameters

None

Return Value

NSM_API_SET_ERROR when the PW port VLAN is not the default for the Ethernet

NSM_API_GET_SUCCESS when the function succeeds

nsm_mpls_set_pw_enet_port_vlan

This function sets the identifier of the VLAN port.

Syntax

```
#include "nsm/mpls/nsm_mpls_vc_snmp.h"
u_int32_t
nsm_mpls_set_pw_enet_port_vlan (struct nsm_master *nm,
                                u_int32_t pw_ix, u_int32_t pw_instance,
                                u_int32_t pw_port_vlan)
```

Input Parameters

`nm`	Pointer to the NSM master structure
`pw_ix`	Index to the virtual circuit table
`pw_instance`	Index to the row in the PW ENET table
`pw_port_vlan`	Port ID

Output Parameters

None

Return Value

NSM_API_SET_ERROR when the PW port VLAN is not the default for the Ethernet

NSM_API_SET_SUCCESS when the function succeeds

nsm_mpls_set_pw_enet_pw_if_index

This function sets the interface index.

Syntax

```
#include "nsm/mpls/nsm_mpls_vc_snmp.h"
u_int32_t
nsm_mpls_set_pw_enet_pw_if_index (struct nsm_master *nm,
                                  u_int32_t pw_ix, u_int32_t pw_instance,
                                  u_int32_t pw_ifindex)
```

Input Parameters

nm	Pointer to the NSM master structure
pw_ix	Index to the virtual circuit table
pw_instance	Index to the row in the PW ENET table
pw_ifindex	Interface index

Output Parameters

None

Return Value

NSM_API_SET_ERROR when the VLAN is not Ethernet tagged

NSM_API_SET_SUCCESS when the function succeeds

nsm_mpls_set_pw_enet_pw_vlan

This function sets the identifier of the ENET VLAN.

Syntax

```
#include "nsm/mpls/nsm_mpls_vc_snmp.h"
u_int32_t
nsm_mpls_set_pw_enet_pw_vlan (struct nsm_master *nm,
                              u_int32_t pw_ix, u_int32_t pw_instance,
                              u_int32_t pw_vlan)
```

Input Parameters

nm	Pointer to the NSM master structure
pw_ix	Index to the virtual circuit table
pw_instance	Index to the row in the PW ENET table
pw_vlan	VLAN ID

Output Parameters

None

Return Value

NSM_API_SET_ERROR when the VLAN ID is not found

© 2015 IP Infusion Inc. Proprietary

NSM_API_SET_SUCCESS when the function succeeds

nsm_mpls_set_pw_enet_row_status

This function sets the row the status.

Syntax

```
#include "nsm/mpls/nsm_mpls_vc_snmp.h"
u_int32_t
nsm_mpls_set_pw_enet_row_status (struct nsm_master *nm,
                                 u_int32_t pw_ix, u_int32_t pw_instance,
                                 u_int32_t row_status)
```

Input Parameters

`nm`	Pointer to the NSM master structure
`pw_ix`	Index to the virtual circuit table
`pw_instance`	Index to the row in the PW ENET table
`row_status`	Row status

Output Parameters

None

Return Value

NSM_API_SET_ERROR when the VLAN is Ethernet tagged and the row status is destroy

NSM_API_SET_SUCCESS when the function succeeds

nsm_mpls_set_pw_enet_storage_type

This function sets the storage type. By default, this value is set to volatile.

Syntax

```
#include "nsm/mpls/nsm_mpls_vc_snmp.h"
u_int32_t
nsm_mpls_set_pw_enet_storage_type (struct nsm_master *nm,
                                   u_int32_t pw_ix, u_int32_t pw_instance,
                                   u_int32_t pw_storage_type)
```

Input Parameters

`nm`	Pointer to the NSM master structure
`pw_ix`	Index to the virtual circuit table
`pw_instance`	Index to the row in the PW ENET table
`pw_storage_type`	The storage type

Output Parameters

None

Return Value

NSM_API_SET_ERROR when the PW storage type is not volatile

NSM_API_SET_SUCCESS when the function succeeds

nsm_mpls_set_pw_enet_vlan_mode

This function sets the VLAN mode value. By default, this value is set to 2 and cannot be modified.

Syntax

```
#include "nsm/mpls/nsm_mpls_vc_snmp.h"
u_int32_t
nsm_mpls_set_pw_enet_vlan_mode (struct nsm_master *nm,
                                u_int32_t pw_ix, u_int32_t pw_instance,
                                u_int32_t pw_vlan_mode)
```

Input Parameters

nm	Pointer to the NSM master structure
pw_ix	Index to the virtual circuit table
pw_instance	Index to the row in the PW ENET table
pw_vlan_mode	VLAN mode

Output Parameters

None

Return Value

NSM_API_SET_ERROR when the VLAN mode is not 2

NSM_API_SET_SUCCESS when the function succeeds

nsm_mpls_set_pw_fcs_retentn_cfg

This function sets the FCS retention configuration for a virtual circuit.

Syntax

```
#include "nsm/mpls/nsm_mpls_vc_snmp.h"
u_int32_t
nsm_mpls_set_pw_fcs_retentn_cfg (struct nsm_master *nm, u_int32_t pw_ix,
                                 u_int32_t retn_cfg);
```

Input Parameters

nm	Pointer to the NSM master structure
pw_ix	Index to the virtual circuit table
retn_cfg	Retention configuration

Output Parameters

None

© 2015 IP Infusion Inc. Proprietary

Return Value

NSM_API_SET_ERROR when the PW index is not valid

NSM_API_SET_SUCCESS when the function succeeds

nsm_mpls_set_pw_frgmt_cfg_size

This function sets the type for a virtual circuit.

Syntax

```
#include "nsm/mpls/nsm_mpls_vc_snmp.h"
u_int32_t
nsm_mpls_set_pw_frgmt_cfg_size (struct nsm_master *nm,
                                u_int32_t pw_ix, u_int32_t cfg_sz);
```

Input Parameters

`nm`	Pointer to the NSM master structure
`pw_ix`	Index to the virtual circuit table
`cfg_sz`	Value

Output Parameters

None

Return Value

NSM_API_SET_ERROR when the PW index is not valid

NSM_API_SET_SUCCESS when the function succeeds

nsm_mpls_set_pw_gen_agi_type

This function sets the attachment group identifier (AGI) type for a virtual circuit.

Syntax

```
#include "nsm/mpls/nsm_mpls_vc_snmp.h"
u_int32_t
nsm_mpls_set_pw_gen_agi_type (struct nsm_master *nm,u_int32_t pw_ix,
                              u_int32_t pw_gen_agi_type)
```

Input Parameters

`nm`	Pointer to the NSM master structure
`pw_ix`	Index to the virtual circuit table
`pw_gen_agi_type`	AGI type

Output Parameters

None

Return Value

NSM_API_SET_ERROR when the PW index is not valid

NSM_API_SET_SUCCESS when the function succeeds

nsm_mpls_set_pw_gen_loc_aii_type

This function sets the local forwarder attachment individual identifier (AII) type for a virtual circuit.

Syntax

```
#include "nsm/mpls/nsm_mpls_vc_snmp.h"
u_int32_t
nsm_mpls_set_pw_gen_loc_aii_type (struct nsm_master *nm,u_int32_t pw_ix,
                                  u_int32_t gen_loc_aii_type)
```

Input Parameters

`nm`	Pointer to the NSM master structure
`pw_ix`	Index to the virtual circuit table
`gen_loc_aii_type`	Local attachment individual identifier type

Output Parameters

None

Return Value

NSM_API_SET_ERROR when the PW index is not valid

NSM_API_SET_SUCCESS when the function succeeds

nsm_mpls_set_pw_gen_rem_aii_type

This function sets the remote forwarder attachment individual identifier (AII) type for a virtual circuit.

Syntax

```
#include "nsm/mpls/nsm_mpls_vc_snmp.h"
u_int32_t
nsm_mpls_set_pw_gen_rem_aii_type (struct nsm_master *nm,u_int32_t pw_ix,
                                  u_int32_t gen_rem_aii_type)
```

Input Parameters

`nm`	Pointer to the NSM master structure
`pw_ix`	Index to the virtual circuit table
`gen_rem_aii_type`	Remote attachment individual identifier type

Output Parameters

None

© 2015 IP Infusion Inc. Proprietary

Return Value

NSM_API_SET_ERROR when the PW index is not valid

NSM_API_SET_SUCCESS when the function succeeds

nsm_mpls_set_pw_grp_attchmt_id

This function sets the group attachment identifier for a virtual circuit.

Syntax

```
#include "nsm/mpls/nsm_mpls_vc_snmp.h"
u_int32_t
nsm_mpls_set_pw_grp_attchmt_id (struct nsm_master *nm,
                                u_int32_t pw_ix, u_int32_t grp_id);
```

Input Parameters

nm	Pointer to the NSM master structure
pw_ix	Index to the virtual circuit table
grp_id	ID value

Output Parameters

None

Return Value

NSM_API_SET_ERROR when the PW index is not valid

NSM_API_SET_SUCCESS when the function succeeds

nsm_mpls_set_pw_hold_prty

This function sets the hold priority for a virtual circuit.

Syntax

```
#include "nsm/mpls/nsm_mpls_vc_snmp.h"
u_int32_t
nsm_mpls_set_pw_hold_prty (struct nsm_master *nm,
                           u_int32_t pw_ix, u_int32_t hold_prty);
```

Input Parameters

nm	Pointer to the NSM master structure
pw_ix	Index to the virtual circuit table
hold_prty	PSN type

Output Parameters

None

Return Value

NSM_API_GET_SUCCESS when the function succeeds

NSM_API_SET_ERROR when the PW index is not valid

nsm_mpls_set_pw_id

This function sets the identifier for a virtual circuit.

Syntax

```
#include "nsm/mpls/nsm_mpls_vc_snmp.h"
u_int32_t
nsm_mpls_set_pw_id (struct nsm_master *nm, u_int32_t pw_ix, u_int32_t pw_id);
```

Input Parameters

nm	Pointer to the NSM master structure
pw_ix	Index to the virtual circuit table
pw_id	ID

Output Parameters

None

Return Value

NSM_API_SET_ERROR when the PW index is not valid

NSM_API_SET_SUCCESS when the function succeeds

nsm_mpls_set_pw_if_ix

This function sets the interface Index to a virtual circuit.

Syntax

```
#include "nsm/mpls/nsm_mpls_vc_snmp.h"
u_int32_t
nsm_mpls_set_pw_if_ix (struct nsm_master *nm, u_int32_t pw_ix, u_int32_t if_ix);
```

Input Parameters

nm	Pointer to the NSM master structure
pw_ix	Index to the virtual circuit table
if_ix	Interface index

Output Parameters

None

Return Value

NSM_API_SET_ERROR when the PW index is not valid

NSM_API_SET_SUCCESS when the function succeeds

© 2015 IP Infusion Inc. Proprietary

nsm_mpls_set_pw_inbd_label

This function sets the inbound label for a virtual circuit.

Syntax

```
#include "nsm/mpls/nsm_mpls_vc_snmp.h"
u_int32_t
nsm_mpls_set_pw_inbd_label (struct nsm_master *nm, u_int32_t pw_ix,
                            u_int32_t in_label);
```

Input Parameters

`nm`	Pointer to the NSM master structure
`pw_ix`	Index to the virtual circuit table
`in_label`	Label

Output Parameters

None

Return Value

NSM_API_SET_ERROR when the PW index is not valid

NSM_API_SET_SUCCESS when the function succeeds

nsm_mpls_set_pw_local_attchmt_id

This function sets the local attachment ID for a virtual circuit.

Syntax

```
#include "nsm/mpls/nsm_mpls_vc_snmp.h"
u_int32_t
nsm_mpls_set_pw_local_attchmt_id (struct nsm_master *nm,
                                  u_int32_t pw_ix, u_int32_t local_id);
```

Input Parameters

`nm`	Pointer to the NSM master structure
`pw_ix`	Index to the virtual circuit table
`local_id`	Attachment ID

Output Parameters

None

Return Value

NSM_API_SET_ERROR when the PW index is not valid

NSM_API_GET_SUCCESS when the function succeeds

nsm_mpls_set_pw_local_capab_advrt

This function sets the local capabilities to be advertised for a virtual circuit.

Syntax

```
#include "nsm/mpls/nsm_mpls_vc_snmp.h"
u_int32_t
nsm_mpls_set_pw_local_capab_advrt (struct nsm_master *nm,
                                   u_int32_t pw_ix, u_int32_t capab);
```

Input Parameters

`nm`	Pointer to the NSM master structure
`pw_ix`	Index to the virtual circuit table
`capab`	Capabilities

Output Parameters

None

Return Value

NSM_API_SET_ERROR when the PW index is not valid

NSM_API_SET_SUCCESS when the function succeeds

nsm_mpls_set_pw_local_grp_id

This function sets the local group identifier for a virtual circuit.

Syntax

```
#include "nsm/mpls/nsm_mpls_vc_snmp.h"
u_int32_t
nsm_mpls_set_pw_local_grp_id (struct nsm_master *nm,
                              u_int32_t pw_ix, u_int32_t local_grp_id);
```

Input Parameters

`nm`	Pointer to the NSM master structure
`pw_ix`	Index to the virtual circuit table
`local_grp_id`	Local group ID

Output Parameters

None

Return Value

NSM_API_SET_ERROR when the PW index is not valid

NSM_API_SET_SUCCESS when the function succeeds

© 2015 IP Infusion Inc. Proprietary

nsm_mpls_set_pw_local_if_mtu

This function sets the local interface MTU for a virtual circuit.

Syntax

```
#include "nsm/mpls/nsm_mpls_vc_snmp.h"
u_int32_t
nsm_mpls_set_pw_local_if_mtu (struct nsm_master *nm,
                              u_int32_t pw_ix, u_int32_t mtu);
```

Input Parameters

`nm`	Pointer to the NSM master structure
`pw_ix`	Index to the virtual circuit table
`mtu`	MTU

Output Parameters

None

Return Value

NSM_API_SET_ERROR when the PW index is not valid

NSM_API_SET_SUCCESS when the function succeeds

nsm_mpls_set_pw_local_if_string

This function sets the local interface string for a virtual circuit.

Syntax

```
#include "nsm/mpls/nsm_mpls_vc_snmp.h"
u_int32_t
nsm_mpls_set_pw_local_if_string (struct nsm_master *nm,
                                 u_int32_t pw_ix, u_int32_t if_string);
```

Input Parameters

`nm`	Pointer to the NSM master structure
`pw_ix`	Index to the virtual circuit table
`if_string`	Value

Output Parameters

None

Return Value

NSM_API_SET_ERROR when the PW index is not valid

NSM_API_SET_SUCCESS when the function succeeds

nsm_mpls_set_pw_mpls_exp_bits

This function sets the type of shim label EXP bit for a virtual circuit.

Syntax

```
#include "nsm/mpls/nsm_mpls_vc_snmp.h"
u_int32_t
nsm_mpls_set_pw_mpls_exp_bits (struct nsm_master *nm,
                               u_int32_t pw_ix, u_int32_t bits);
```

Input Parameters

`nm`	Pointer to the NSM master structure
`pw_ix`	Index to the virtual circuit table
`bits`	Bits; the only permitted value is 0

Output Parameters

None

Return Value

NSM_API_SET_ERROR when the PW index is not valid

NSM_API_SET_SUCCESS when the function succeeds

nsm_mpls_set_pw_mpls_exp_bits_mode

This function sets the type of shim label EXP bit mode for a virtual circuit.

Syntax

```
#include "nsm/mpls/nsm_mpls_vc_snmp.h"
u_int32_t
nsm_mpls_set_pw_mpls_exp_bits_mode (struct nsm_master *nm,
                                    u_int32_t pw_ix, u_int32_t mode);
```

Input Parameters

`nm`	Pointer to the NSM master structure
`pw_ix`	Index to the virtual circuit table
`mode`	Mode; the only permitted value is `NSM_MPLS_PW_OUTER_TUNNEL` defined in `nsm/mpls/nsm_mpls_vc_api.h`

Output Parameters

None

Return Value

NSM_API_SET_ERROR when the PW index is not valid

NSM_API_SET_SUCCESS when the function succeeds

© 2015 IP Infusion Inc. Proprietary

nsm_mpls_set_pw_mpls_lcl_ldp_entty_ix

This function sets the local LDP entity index for a virtual circuit.

Syntax

```
#include "nsm/mpls/nsm_mpls_vc_snmp.h"
u_int32_t
nsm_mpls_set_pw_mpls_lcl_ldp_entty_ix (struct nsm_master *nm,
                                       u_int32_t pw_ix, u_int32_t entity_ix);
```

Input Parameters

`nm`	Pointer to the NSM master structure
`pw_ix`	Index to the virtual circuit table
`entity_ix`	Local LDP entity index

Output Parameters

None

Return Value

NSM_API_SET_ERROR when the PW index is not valid

NSM_API_SET_SUCCESS when the function succeeds

nsm_mpls_set_pw_mpls_lcl_ldp_id

This function sets the local identifier of the LDP entity which creates the PW for a virtual circuit.

Syntax

```
#include "nsm/mpls/nsm_mpls_vc_snmp.h"
u_int32_t
nsm_mpls_set_pw_mpls_lcl_ldp_id (struct nsm_master *nm, u_int32_t pw_ix,
                                 struct ldp_id *lcl_ldp_id);
```

Input Parameters

`nm`	Pointer to the NSM master structure
`pw_ix`	Index to the virtual circuit table
`lcl_ldp_id`	Local LDP identifier

Output Parameters

None

Return Value

NSM_API_SET_ERROR when the PW index is not valid

NSM_API_SET_SUCCESS when the function succeeds

nsm_mpls_set_pw_mpls_mpls_type

This function sets the outer tunnel type for a virtual circuit.

Syntax

```
#include "nsm/mpls/nsm_mpls_vc_snmp.h"
u_int32_t
nsm_mpls_set_pw_mpls_mpls_type (struct nsm_master *nm,
                                u_int32_t pw_ix, u_int32_t mpls_type);
```

Input Parameters

`nm`	Pointer to the NSM master structure
`pw_ix`	Index to the virtual circuit table
`mpls_type`	Type; one of these constants from `nsm/mpls/nsm_mpls_vc_api.h`:
`MPLSTE`	Outer tunnel set up by MPLS-TE
`MPLSNONTE`	Outer tunnel set up by LDP
`PWONLY`	No outer tunnel label; static provisioning without an MPLS tunnel

Output Parameters

None

Return Value

NSM_API_SET_ERROR when the PW index is not valid

NSM_API_SET_SUCCESS when the function succeeds

nsm_mpls_set_pw_mpls_outbd_if_ix

This function sets the outbound interface index for a virtual circuit.

Syntax

```
#include "nsm/mpls/nsm_mpls_vc_snmp.h"
u_int32_t
nsm_mpls_set_pw_mpls_outbd_if_ix (struct nsm_master *nm,
                                  u_int32_t pw_ix, u_int32_t if_ix);
```

Input Parameters

`nm`	Pointer to the NSM master structure
`pw_ix`	Index to the virtual circuit table
`if_ix`	Outbound interface index

Output Parameters

None

Return Value

NSM_API_SET_ERROR when the PW index is not valid

© 2015 IP Infusion Inc. Proprietary

NSM_API_SET_SUCCESS when the function succeeds

nsm_mpls_set_pw_mpls_outbd_lsr_xc_ix

This function sets the outbound tunnel cross connect index for a virtual circuit.

Syntax

```
#include "nsm/mpls/nsm_mpls_vc_snmp.h"
u_int32_t
nsm_mpls_set_pw_mpls_outbd_lsr_xc_ix (struct nsm_master *nm,
                                      u_int32_t pw_ix, u_int32_t xc_ix);
```

Input Parameters

nm	Pointer to the NSM master structure
pw_ix	Index to the virtual circuit table
xc_ix	Outbound tunnel index

Output Parameters

None

Return Value

NSM_API_SET_ERROR when the PW index is not valid

NSM_API_SET_SUCCESS when the function succeeds

nsm_mpls_set_pw_mpls_outbd_tnl_ix

This function sets the outbound tunnel index for a virtual circuit.

Syntax

```
#include "nsm/mpls/nsm_mpls_vc_snmp.h"
u_int32_t
nsm_mpls_set_pw_mpls_outbd_tnl_ix (struct nsm_master *nm,
                                   u_int32_t pw_ix, u_int32_t tnl_ix);
```

Input Parameters

nm	Pointer to the NSM master structure
pw_ix	Index to the virtual circuit table
tnl_ix	Outbound tunnel index

Output Parameters

None

Return Value

NSM_API_SET_ERROR when the PW index is not valid

NSM_API_SET_SUCCESS when the function succeeds

nsm_mpls_set_pw_mpls_outbd_tnl_lcl_lsr

This function sets the outbound tunnel LSR identifier for a virtual circuit.

Syntax

```
#include "nsm/mpls/nsm_mpls_vc_snmp.h"
u_int32_t
nsm_mpls_set_pw_mpls_outbd_tnl_lcl_lsr (struct nsm_master *nm,
                                        u_int32_t pw_ix,
                                        struct pal_in4_addr *lcl_lsr);
```

Input Parameters

nm	Pointer to the NSM master structure
pw_ix	Index to the virtual circuit table
lcl_lsr	ID

Output Parameters

None

Return Value

NSM_API_SET_ERROR when the PW index is not valid

NSM_API_SET_SUCCESS when the function succeeds

nsm_mpls_set_pw_mpls_outbd_tnl_peer_lsr

This function sets the outbound tunnel peer LSR address for a virtual circuit.

Syntax

```
#include "nsm/mpls/nsm_mpls_vc_snmp.h"
u_int32_t
nsm_mpls_set_pw_mpls_outbd_tnl_peer_lsr (struct nsm_master *nm,
                                         u_int32_t pw_ix,
                                         struct pal_in4_addr *peer_lsr);
```

Input Parameters

nm	Pointer to the NSM master structure
pw_ix	Index to the virtual circuit table
peer_lsr	LSR address

Output Parameters

None

Return Value

NSM_API_SET_ERROR when the PW index is not valid

NSM_API_SET_SUCCESS when the function succeeds

© 2015 IP Infusion Inc. Proprietary

nsm_mpls_set_pw_mpls_ttl

This function sets the TTL on a PW shim for a virtual circuit.

Syntax

```
#include "nsm/mpls/nsm_mpls_vc_snmp.h"
u_int32_t
nsm_mpls_set_pw_mpls_ttl (struct nsm_master *nm,
                          u_int32_t pw_ix, u_int32_t ttl);
```

Input Parameters

`nm`	Pointer to the NSM master structure
`pw_ix`	Index to the virtual circuit table
`ttl`	TTL value; the only permitted value is `NSM_MPLS_PW_TTL_VALUE` defined in `nsm/mpls/nsm_mpls_vc_api.h`

Output Parameters

None

Return Value

NSM_API_SET_ERROR when the PW index is not valid

NSM_API_SET_SUCCESS when the function succeeds

nsm_mpls_set_pw_name

This function sets the name for a virtual circuit.

Syntax

```
#include "nsm/mpls/nsm_mpls_vc_snmp.h"
u_int32_t
nsm_mpls_set_pw_name (struct nsm_master *nm,
                      u_int32_t pw_ix, char *name);
```

Input Parameters

`nm`	Pointer to the NSM master structure
`pw_ix`	Index to the virtual circuit table
`name`	Name

Output Parameters

None

Return Value

NSM_API_SET_ERROR when the PW index is not valid

NSM_API_SET_SUCCESS when the function succeeds

nsm_mpls_set_pw_notify_rate

This function sets the notification rate.

Syntax

```
#include "nsm/mpls/nsm_mpls_vc_api.h"
u_int32_t
nsm_mpls_set_pw_notify_rate (struct nsm_mpls *nmpls, u_int32_t val);
```

Input Parameters

nmpls	NSM MPLS pointer
val	Notification rate

Output Parameters

None

Return Value

NSM_API_SET_ERROR when the NSM MPLS pointer is NULL

NSM_API_SET_SUCCESS when the function succeeds

nsm_mpls_set_pw_oam_enable

This function enables or disables PW OAM for a virtual circuit.

Syntax

```
#include "nsm/mpls/nsm_mpls_vc_snmp.h"
u_int32_t
nsm_mpls_set_pw_oam_enable (struct nsm_master *nm,u_int32_t pw_ix, u_int32_t oam_en)
```

Input Parameters

nm	Pointer to the NSM master structure
pw_ix	Index to the virtual circuit table
oam_en	PAL_TRUE or PAL_FALSE

Output Parameters

None

Return Value

NSM_API_SET_ERROR when the PW index is not valid

NSM_API_SET_SUCCESS when the function succeeds

nsm_mpls_set_pw_outbd_label

This function sets the outbound label for a virtual circuit.

© 2015 IP Infusion Inc. Proprietary

Syntax

```
#include "nsm/mpls/nsm_mpls_vc_snmp.h"
u_int32_t
nsm_mpls_set_pw_outbd_label (struct nsm_master *nm, u_int32_t pw_ix,
                             u_int32_t out_label);
```

Input Parameters

`nm`	Pointer to the NSM master structure
`pw_ix`	Index to the virtual circuit table
`out_label`	Label

Output Parameters

None

Return Value

NSM_API_SET_ERROR when the PW index is not valid

NSM_API_SET_SUCCESS when the function succeeds

nsm_mpls_set_pw_owner

This function sets the owner for a virtual circuit.

Syntax

```
#include "nsm/mpls/nsm_mpls_vc_snmp.h"
u_int32_t
nsm_mpls_set_pw_owner (struct nsm_master *nm, u_int32_t pw_ix,
                       u_int32_t pw_owner);
```

Input Parameters

`nm`	Pointer to the NSM master structure
`pw_ix`	Index to the virtual circuit table
`pw_owner`	Owner; one of these constants from `nsm/mpls/nsm_mpls_vc.h`:

`PW_OWNER_GEN_FEC_SIGNALING`

`PW_OWNER_PWID_FEC_SIGNALING`

`PW_OWNER_MANUAL`

Output Parameters

None

Return Value

NSM_API_SET_ERROR when the PW index is not valid

NSM_API_GET_SUCCESS when the function succeeds

nsm_mpls_set_pw_peer_addr

This function sets the peer address for a virtual circuit.

Syntax

```
#include "nsm/mpls/nsm_mpls_vc_snmp.h"
u_int32_t
nsm_mpls_set_pw_peer_addr (struct nsm_master *nm, u_int32_t pw_ix,
                           struct pal_in4_addr *peer_addr);
```

Input Parameters

nm	Pointer to the NSM master structure
pw_ix	Index to the virtual circuit table
peer_addr	Address

Output Parameters

None

Return Value

NSM_API_SET_ERROR when the PW index is not valid

NSM_API_SET_SUCCESS when the function succeeds

nsm_mpls_set_pw_peer_attchmt_id

This function sets the peer attachment ID for a virtual circuit.

Syntax

```
#include "nsm/mpls/nsm_mpls_vc_snmp.h"
u_int32_t
nsm_mpls_set_pw_peer_attchmt_id (struct nsm_master *nm,
                                 u_int32_t pw_ix, u_int32_t peer_id);
```

Input Parameters

nm	Pointer to the NSM master structure
pw_ix	Index to the virtual circuit table
peer_id	Peer ID

Output Parameters

None

Return Value

NSM_API_SET_ERROR when the PW index is not valid

NSM_API_SET_SUCCESS when the function succeeds

© 2015 IP Infusion Inc. Proprietary

nsm_mpls_set_pw_psn_type

This function sets the PSN type for a virtual circuit.

Syntax

```
#include "nsm/mpls/nsm_mpls_vc_snmp.h"
u_int32_t
nsm_mpls_set_pw_psn_type (struct nsm_master *nm, u_int32_t pw_ix,
                          u_int32_t psn_type);
```

Input Parameters

`nm`	Pointer to the NSM master structure
`pw_ix`	Index to the virtual circuit table
`psn_type`	PSN type

Output Parameters

None

Return Value

NSM_API_SET_ERROR when the PW index is not valid

NSM_API_GET_SUCCESS when the function succeeds

nsm_mpls_set_pw_row_status

This function sets the row status for a virtual circuit.

Syntax

```
#include "nsm/mpls/nsm_mpls_vc_snmp.h"
u_int32_t
nsm_mpls_set_pw_row_status (struct nsm_master *nm, u_int32_t pw_ix,
                            u_int32_t row_status);
```

Input Parameters

`nm`	Pointer to the NSM master structure
`pw_ix`	Index to the virtual circuit table
`row_status`	Status

Output Parameters

None

Return Value

NSM_API_SET_ERROR when the PW index is not valid

NSM_API_GET_SUCCESS when the function succeeds

nsm_mpls_set_pw_setup_prty

This function sets the setup priority for a virtual circuit.

Syntax

```
#include "nsm/mpls/nsm_mpls_vc_snmp.h"
u_int32_t
nsm_mpls_set_pw_setup_prty (struct nsm_master *nm, u_int32_t pw_ix,
                            u_int32_t setup_prty);
```

Input Parameters

`nm`	Pointer to the NSM master structure
`pw_ix`	Index to the virtual circuit table
`setup_prty`	Priority

Output Parameters

None

Return Value

NSM_API_SET_ERROR when the PW index is not valid

NSM_API_SET_SUCCESS when the function succeeds

nsm_mpls_set_pw_st_type

This function sets the storage type for a virtual circuit.

Syntax

```
#include "nsm/mpls/nsm_mpls_vc_snmp.h"
u_int32_t nsm_mpls_set_pw_st_type (struct nsm_master *nm, u_int32_t pw_ix,
                                   u_int32_t st_type);
```

Input Parameters

`nm`	Pointer to the NSM master structure
`pw_ix`	Index to the virtual circuit table
`st_type`	Storage type

Output Parameters

None

Return Value

NSM_API_SET_ERROR when the PW index is not valid

NSM_API_SET_SUCCESS when the function succeeds

nsm_mpls_set_pw_type

This function sets the type for a virtual circuit.

© 2015 IP Infusion Inc. Proprietary

Syntax

```
#include "nsm/mpls/nsm_mpls_vc_snmp.h"
u_int32_t
nsm_mpls_set_pw_type (struct nsm_master *nm, u_int32_t pw_ix,
                      u_int32_t pw_type);
```

Input Parameters

`nm`	Pointer to the NSM master structure
`pw_ix`	Index to the virtual circuit table
`pw_type`	Type

Output Parameters

None

Return Value

NSM_API_SET_ERROR when the PW index is not valid

NSM_API_GET_SUCCESS when the function succeeds

nsm_mpls_set_pw_up_dn_notify

This function enables or disables up down notification for the virtual circuit table.

Syntax

```
#include "nsm/mpls/nsm_mpls_vc_api.h"
u_int32_t
nsm_mpls_set_pw_up_dn_notify (struct nsm_mpls *nmpls, int val);
```

Input Parameters

`nmpls`	NSM MPLS pointer
`val`	Notification setting

Output Parameters

None

Return Value

NSM_API_SET_ERROR when the NSM MPLS pointer is NULL

NSM_API_SET_SUCCESS when the function succeeds

© 2015 IP Infusion Inc. Proprietary

CHAPTER 18 Static VPLS API

Static configuration support for VPLS uses VPLS mesh VC setup using LDP signaling as well as the following:

- Label information: When a VPLS peer is added to a VPLS instance, NSM forwards the peer information to LDP to request label information. Label information consists of incoming label, outgoing label, incoming interface, and outgoing interface data. Static FIB Entry is configured with label information and a `vpls_peer` → `vc_fib` association is created for the static information.
- Forwarding information: You can specify tunnel information for selecting the tunnel (that is, binding the VC to the LSP). Tunnel data consists of `tunnel-name` or `tunnel-id`. This information is checked against the entries in the FTN table and the resulting tunnel is used for adding a FIB entry using `nsm_vpls_vc_fib_add` API. The FTN table is looked up with FEC (`vpls_peer` → `peer_addr`) and the first entry (`ftn` → `head`) is the t`unnel-id`.

Functions

The following subsection includes the APIs for static VPLS.

nsm_vpls_vc_fib_entry_add_process

This call is used to add a VPLS mesh or spoke VC static FIB entry.

Syntax

```
nsm_vpls_vc_fib_entry_add_process(u_int32_t vr_id,
                                  char* vpls_id_str,u_int8_t vc_style,
                                  char *egress,char* in_lbl,
                                  char* out_if, char* out_lbl)
```

Input Parameters

`vr_id`	VR identifier
`vpls_id_str`	VPLS identifier
`vc_style`	Set VC style as mesh or spoke
`egress`	Peer address
`in_lbl`	Incoming label
`out_if`	Provider-facing interface
`out_lbl`	Outgoing label

Output Parameters

none

nsm_vpls_vc_fib_entry_delete_process

This call is used to delete a Static VPLS FIB entry.

Syntax

```
nsm_vpls_vc_fib_entry_delete_process (u_int32_t vr_id,
                                      char* vpls_id_str, u_int8_t vc_style,
                                      char *egress, char* in_lbl_str,
                                      char* if_out, char* out_lbl_str,
                                      u_char params)
```

Input Parameters

`vr_id`	VR identifier
`vpls_id_str`	VPLS identifier
`vc_style`	Set VC style as mesh or spoke
`egress`	Peer address
`in_lbl_str`	Incoming label
`if_out`	Provider-facing interface
`out_lbl_str`	Outgoing label
`params`	Identify whether all the parameters are provided for deleting the entry

Output Parameters

None

nsm_vpls_mesh_peer_ipv4_add_cli

This call is used to add a VPLS mesh peer.

Syntax

```
nsm_vpls_mesh_peer_ipv4_add_cli(u_int32_t vr_id,
                                u_int32_t vpls_id, char *sz_peer,
                                u_char mapping_type,
                                char *tunnel_info, u_int8_t fec_type_vc,
                                char *sz_mode, char *sec_peer)
```

Input Parameters

`vr_id`	VR identifier
`vpls_id_str`	VPLS identifier
`sz_peer`	Peer address
`mapping_type`	Mapping type
`tunnel_info`	Tunnel name and tunnel ID information
`fec_type_vc`	Identify VC type
`sz_mode`	Specify mode
`sec_peer`	Secondary peer address

Output Parameter

None

© 2015 IP Infusion Inc. Proprietary

CHAPTER 19 MPLS-TP Command API

The chapter describes the functions for basic MPLS-TP configuration.

nsm_api_itut_mpls_tp_global_config

This function sets the ITU-T country code, carrier code, and node identifier.

This function is called by the `mpls-tp itut cc icc node-id` command.

Syntax

```
int
nsm_api_itut_mpls_tp_global_config (u_int32_t vr_id, u_char *cc,
                                    u_char *icc, u_int32_t nid)
```

Input Parameters

`vr_id`	Virtual router instance identifier
`cc`	Country code: two upper-case letters (A-Z)
`icc`	Carrier code: 1-6 upper-case letters (A-Z) or digits (0-9)
`nid`	Node identifier

Output Parameters

None

Return Values

NSM_API_SET_ERR_VR_NOT_EXIST when the virtual router does not exist

NSM_API_SET_ERR_CC_NAME_INVALID when `cc` is not valid

NSM_API_SET_ERR_ICC_NAME_INVALID when `icc` is not valid

NSM_API_SET_ERR_MPLS_TP_NODE_ID_INVALID when `nid` is not valid

NSM_API_SET_ERR_MPLS_TP_ACTIVE_TUNNEL_EXISTS when a tunnel exists

NSM_API_SET_SUCCESS when the function succeeds

nsm_api_itut_mpls_tp_global_unconfig

This function removes the ITU-T country code, carrier code, and node identifier.

This function is called by the `no mpls-tp itut cc icc node-id` command.

Syntax

```
int
nsm_api_itut_mpls_tp_global_unconfig (u_int32_t vr_id, u_char *cc,
                                      u_char *icc, u_int32_t nid)
```

Input Parameters

`vr_id`	Virtual router instance identifier
`cc`	Country code: two upper-case letters (A-Z)
`icc`	Carrier code: 1-6 alphabetic (A-Z) or numeric (0-9) characters
`nid`	Node identifier

Output Parameters

None

Return Values

NSM_API_SET_ERR_VR_NOT_EXIST when the virtual router does not exist

NSM_API_SET_ERR_CC_NAME_INVALID when `cc` is not valid

NSM_API_SET_ERR_ICC_NAME_INVALID when `icc` is not valid

NSM_API_SET_ERR_MPLS_TP_NODE_ID_INVALID when `nid` is not valid

NSM_API_SET_ERR_MPLS_TP_ACTIVE_TUNNEL_EXISTS when a tunnel exists

NSM_API_SET_ERR_MPLS_TP_ID_NOT_MATCH when `cc` and/or `icc` do not match configured values

NSM_API_SET_ERR_MPLS_TP_ID_CONFIG_NOT_EXISTS when

NSM_API_SET_SUCCESS when the function succeeds

nsm_api_itut_mpls_tp_tnl_config

This function creates an ITU-T MPLS-TP tunnel.

This function is called by the `mpls-tp tunnel <1-65535> source CC-NAME ICC-NAME A.B.C.D destination CC-NAME ICC-NAME A.B.C.D` command.

Syntax

```
int
nsm_api_itut_mpls_tp_tnl_config (u_int32_t vr_id, u_int32_t tnl_id,
                                 u_char *src_cc, u_char *src_icc, u_int32_t src_nid,
                                 u_char *dst_cc, u_char *dst_icc, u_int32_t dst_nid,
                                 struct mpls_tp_tnl **tnl)
```

Input Parameters

`vr_id`	Virtual router instance identifier
`tnl_id`	Tunnel identifier
`src_cc`	Source country code: two upper-case letters (A-Z)
`src_icc`	Source carrier code: 1-6 alphabetic (A-Z) or numeric (0-9) characters
`src_nid`	Source node identifier
`dst_cc`	Destination country code: two upper-case letters (A-Z)
`dst_icc`	Destination carrier code: 1-6 alphabetic (A-Z) or numeric (0-9) characters
`dst_nid`	Destination node identifier

© 2015 IP Infusion Inc. Proprietary

Output Parameters

`tnl`	MPLS-TP tunnel structure

Return Values

NSM_API_SET_ERR_SRC_CC_NAME_INVALID when `src_cc` is not valid

NSM_API_SET_ERR_SRC_ICC_NAME_INVALID when `src_icc` is not valid

NSM_API_SET_ERR_DST_CC_NAME_INVALID when `dst_cc` is not valid

NSM_API_SET_ERR_DST_ICC_NAME_INVALID when `dst_icc` is not valid

NSM_API_SET_ERR_VR_NOT_EXIST when the virtual router does not exist

NSM_API_SET_ERR_MPLS_TP_ID_CONFIG_NOT_EXISTS when the MPLS-TP identifiers are not configured

NSM_API_SET_ERR_MPLS_TP_TNL_SRC_DST_SAME when the tunnel source and destination identifiers are the same

NSM_API_SET_ERROR when there is an internal error

NSM_API_SET_ERR_MPLS_TP_TNL_DUPLICATE_TNL_ID when there is another tunnel with the same identifier that originates from the same node for a different destination

NSM_API_SET_SUCCESS when the function succeeds

nsm_api_itut_mpls_tp_tnl_unconfig

This function removes an ITU-T MPLS-TP tunnel.

This function is called by the `no mpls-tp tunnel <1-65535> source CC-NAME ICC-NAME A.B.C.D destination CC-NAME ICC-NAME A.B.C.D` command.

Syntax

```
int
nsm_api_itut_mpls_tp_tnl_unconfig (u_int32_t vr_id, u_int32_t tnl_id,
                                   u_char *src_cc, u_char *src_icc, u_int32_t src_nid,
                                   u_char *dst_cc, u_char *dst_icc, u_int32_t dst_nid)
```

Input Parameters

`vr_id`	Virtual router instance identifier
`tnl_id`	Tunnel identifier
`src_cc`	Source country code: two upper-case letters (A-Z)
`src_icc`	Source carrier code: 1-6 alphabetic (A-Z) or numeric (0-9) characters
`src_nid`	Source node identifier
`dst_cc`	Destination country code: two upper-case letters (A-Z)
`dst_icc`	Destination carrier code: 1-6 alphabetic (A-Z) or numeric (0-9) characters
`dst_nid`	Destination node identifier

Output Parameters

None

Return Values

NSM_API_SET_ERR_VR_NOT_EXIST when the virtual router does not exist

NSM_API_SET_ERR_MPLS_TP_TNL_SRC_DST_SAME when the tunnel source and destination identifiers are the same

NSM_API_SET_ERR_MPLS_TP_TNL_DOES_NOT_EXIST when the tunnel does not exist

NSM_API_SET_ERR_MPLS_TP_TNL_ASSOC_EXIST when the tunnel is associated with another tunnel

NSM_API_SET_ERR_MPLS_TP_TNL_SERVICE_MAP_EXIST when the tunnel is mapped to at least one service

NSM_API_SET_ERR_MPLS_TP_TNL_HAS_MEG_CONFIG when the tunnel has an associated MEG

NSM_API_SET_ERROR when there is an internal error

NSM_API_SET_SUCCESS when the function succeeds

nsm_api_mpls_tp_if_set

This function configures the MPLS-TP capability on the specified interface.

This function is called by the `mpls-tp provider-interface` command.

Syntax

```
int
nsm_api_mpls_tp_if_set (u_int32_t vr_id, u_int32_t ifindex,
                        u_char *local_str);
```

Input Parameters

`vr_id`	The virtual router (VR) instance ID
`ifindex`	The outgoing interface index
`local_str`	The name of the interface

Output Parameters

None

Return Values

NSM_API_SET_SUCCESS when the function executed properly

NSM_API_SET_ERROR when an invalid value was entered

NSM_API_SET_ERR_CANT_SET_GMPLS_IF_TO_MPLS_TP when GMPLS is already active; MPLS-TP configuration not allowed on this interface

NSM_API_SET_ERR_IF_NOT_EXIST when the interface does not exist

NSM_API_SET_ERR_IF_TYPE_NOT_DATA when the interface type is not Data or Data-Control

NSM_API_SET_ERR_INVALID_DATA_LINK when the data link is invalid

NSM_API_SET_ERR_INVALID_IF when the interface is invalid

NSM_API_SET_ERR_INVALID_IF_ID when the Interface ID is invalid

NSM_API_SET_ERR_INVALID_IF_TYPE when the interface type is invalid

NSM_API_SET_ERR_LINKID_EXIST when link creation failed; link ID should be unique within data link local-link-id, TE local-link-id and cc-id

 © 2015 IP Infusion Inc. Proprietary

NSM_API_SET_ERR_MPLS_TP_CONF_ALREADY when there is a duplicate entry; MPLS-TP is already configured

NSM_API_SET_ERR_VR_NOT_EXIST when the virtual router does not exist

NSM_ERR_LS_ON_LOOPBACK when unable to enable label switching on loopback

NSM_FAILURE when the MPLS-TP provider interface is not set properly

nsm_api_mpls_tp_if_unset

This function resets (unconfigures) the MPLS-TP capability on an interface.

This function is called by the `no mpls-tp provider-interface` CLI command.

Syntax

```
int
nsm_api_mpls_tp_if_unset (u_int32_t vr_id,u_int32_t ifindex);
```

Input Parameters

`vr_id`	The virtual router (VR) instance ID
`ifindex`	The outgoing interface index

Output Parameters

None

Return Values

NSM_API_SET_SUCCESS when the function executed properly

NSM_API_SET_ERR_IF_NOT_EXIST when the interface does not exist

NSM_API_SET_ERR_VR_NOT_EXIST when the virtual router does not exist

NSM_API_UNSET_ERR_IF_NOT_MPLS_TP_CAP when the interface was not configured as MPLS-TP Provider

NSM_ERR_LS_ON_LOOPBACK when unable to enable label switching on loopback

NSM_FAILURE when unable to disable label switching

NSM_MPLS_TP_FTN_ILM_EXISTS when MPLS-TP is active on this interface; cannot be unset

nsm_api_mpls_tp_global_config

This function sets the ITEF global identifier and node identifier.

This function is called by the `mpls-tp global-id node-id` command.

Syntax

```
int
nsm_api_mpls_tp_global_config (u_int32_t vr_id, u_int32_t gid,
                               u_int32_t nid);
```

Input Parameters

`vr_id`	Virtual router instance identifier
`gid`	Global identifier
`nid`	Node identifier

Output Parameters

None

Return Values

NSM_API_SET_SUCCESS when the function executed properly

NSM_API_SET_ERR_MPLS_TP_NODE_ID_INVALID when the MPLS-TP Node ID is invalid

NSM_API_SET_ERR_MPLS_TP_ACTIVE_TUNNEL_EXISTS when the MPLS-TP Tunnel or VC exists; cannot be updated or deleted

NSM_API_SET_ERR_VR_NOT_EXIST when the virtual router does not exist

nsm_api_mpls_tp_global_unconfig

This function removes the ITEF global identifier and node identifier.

This function is called by the `no mpls-tp global-id node-id` CLI command.

Syntax

```
int
nsm_api_mpls_tp_global_unconfig (u_int32_t vr_id, u_int32_t gid,
                                 u_int32_t nid);
```

Input Parameters

`vr_id`	Virtual router instance identifier
`gid`	Global identifier
`nid`	Node identifier

Output Parameters

None

Return Values

NSM_API_SET_SUCCESS when the function executed properly

NSM_API_SET_ERR_MPLS_TP_ACTIVE_TUNNEL_EXISTS when the MPLS-TP Tunnel or VC exists; cannot be updated or deleted

NSM_API_SET_ERR_MPLS_TP_ID_CONFIG_NOT_EXISTS when the MPLS-TP identifiers are not configured

NSM_API_SET_ERR_MPLS_TP_IDENTIFIERS_INVALID when the MPLS-TP identifiers are invalid

nsm_api_mpls_tp_tnl_config

This function creates an IETF MPLS-TP tunnel.

This function is called by the `mpls-tp tunnel source destination` command.

Syntax

```
int
nsm_api_mpls_tp_tnl_config (u_int32_t vr_id, u_int32_t tnl_id,
                            u_int32_t src_gid, u_int32_t src_nid,
```

 © 2015 IP Infusion Inc. Proprietary

```
                                u_int32_t dst_gid, u_int32_t dst_nid,
                                struct mpls_tp_tnl **tnl);
```

Input Parameters

`vr_id`	The virtual router (VR) instance ID
`tnl_id`	The tunnel ID
`src_gid`	The tunnel source Global ID
`src_nid`	The tunnel source Node ID
`dst_gid`	The tunnel destination Global ID
`dst_nid`	The tunnel destination Node ID

Output Parameters

`tnl`	MPLS-TP tunnel structure

Return Values

NSM_API_SET_SUCCESS when the function executed properly

NSM_API_SET_ERROR when there is an internal error

NSM_API_SET_ERR_VR_NOT_EXIST when the virtual router does not exist

NSM_API_SET_ERR_MPLS_TP_ID_CONFIG_NOT_EXISTS when the MPLS-TP identifiers are not configured

NSM_API_SET_ERR_MPLS_TP_TNL_SRC_DST_SAME when the tunnel source and destination identifiers are the same

NSM_API_SET_ERR_MPLS_TP_TNL_DUPLICATE_TNL_ID when there is another tunnel with the same identifier that originates from the same node for a different destination

nsm_api_mpls_tp_tnl_unconfig

This function removes an IETF MPLS-TP tunnel.

This function is called by the `no mpls-tp tunnel source destination` command.

Syntax

```
int
int nsm_api_mpls_tp_tnl_unconfig (u_int32_t vr_id, u_int32_t tnl_id,
                                  u_int32_t src_gid, u_int32_t src_nid,
                                  u_int32_t dst_gid, u_int32_t dst_nid);
```

Input Parameters

`vr_id`	The virtual router (VR) instance ID
`tnl_id`	The tunnel ID
`src_gid`	The tunnel source Global ID
`src`	The tunnel source Node ID
`dst_gid`	The tunnel destination Global ID
`dst_nid`	The tunnel destination Node ID

Output Parameters

None

Return Values

NSM_API_SET_SUCCESS when the function executed properly

NSM_API_SET_ERROR when there is an internal error

NSM_API_SET_ERR_MPLS_TP_TNL_ASSOC_EXIST when the tunnel is associated with another tunnel

NSM_API_SET_ERR_MPLS_TP_TNL_DOES_NOT_EXIST when the tunnel does not exist

NSM_API_SET_ERR_MPLS_TP_TNL_HAS_MEG_CONFIG when the tunnel has an associated MEG

NSM_API_SET_ERR_MPLS_TP_TNL_SERVICE_MAP_EXIST when the tunnel is mapped to at least one service

NSM_API_SET_ERR_MPLS_TP_TNL_SRC_DST_SAME when the tunnel source and destination identifiers are the same

NSM_API_SET_ERR_VR_NOT_EXIST when the virtual router does not exist

nsm_api_mpls_tp_tnl_name_config

This function validates the tunnel name and updates the same in the tunnel data structure. The tunnel name must be unique; duplicating names is not allowed.

This function is called by the `tunnel-name` command.

Syntax

```
int
nsm_api_mpls_tp_tnl_name_config (u_int32_t vr_id, char *tnl_name,
                                 struct mpls_tp_tnl *tp_tnl);
```

Input Parameters

`vr_id`	The virtual router (VR) instance ID
`*tnl_name`	The character pointer to the tunnel name
`*tp_tnl`	The pointer to the MPLS-TP tunnel structure

Output Parameters

None

Return Values

NSM_API_SET_SUCCESS when the function executed properly

NSM_API_SET_ERROR when an invalid value was entered

NSM_API_SET_ERR_MPLS_TP_TNL_ASSOC_EXIST when the MPLS-TP tunnel is associated with another tunnel

NSM_API_SET_ERR_MPLS_TP_TNL_HAS_MEG_CONFIG when the MPLS-TP tunnel has MEG association

NSM_API_SET_ERR_MPLS_TP_TNL_NAME_IN_USE when the tunnel name is already assigned to another MPLS-TP tunnel

NSM_API_SET_ERR_MPLS_TP_TNL_SERVICE_MAP_EXIST when the MPLS-TP tunnel is mapped to at least one service

NSM_API_SET_ERR_VR_NOT_EXIST when the virtual router does not exist

 © 2015 IP Infusion Inc. Proprietary

nsm_api_mpls_tp_tnl_name_unconfig

This function removes the tunnel name configuration from the tunnel:

- no associated tunnel
- no dependent entries

This function is called by the `no tunnel-name` command.

Syntax

```
int
nsm_api_mpls_tp_tnl_name_unconfig (u_int32_t vr_id,
                                   struct mpls_tp_tnl *tp_tnl);
```

Input Parameters

`vr_id`	The virtual router (VR) instance ID
`*tp_tnl`	The pointer to the MPLS-TP tunnel structure

Output Parameters

None

Return Values

NSM_API_SET_SUCCESS when the function executed properly

NSM_API_SET_ERR_MPLS_TP_TNL_ASSOC_EXIST when the MPLS-TP tunnel is associated with another tunnel

NSM_API_SET_ERR_MPLS_TP_TNL_HAS_MEG_CONFIG when the MPLS-TP tunnel has MEG association

NSM_API_SET_ERR_MPLS_TP_TNL_NAME_NOT_CONFIGURED when the tunnel name was not previously configured

NSM_API_SET_ERR_MPLS_TP_TNL_SERVICE_MAP_EXIST when the MPLS-TP tunnel is mapped to at least one service

NSM_API_SET_ERR_VR_NOT_EXIST when the virtual router does not exist

nsm_api_mpls_tp_tnl_assoc_config_validate

This function associates a forward tunnel with a reverse tunnel. It binds both tunnels and handles the FSM for both the forward and the reverse tunnels.

This function is called by the `mpls-tp associate fwd-tunnel rev-tunnel` command.

Syntax

```
int
nsm_api_mpls_tp_tnl_assoc_config_validate (u_int32_t vr_id,
                                           char *fwd_tnl_name,
                                           char *rev_tnl_name);
```

Input Parameters

`vr_id`	The virtual router (VR) instance ID
`*fwd_tnl_name`	The name of the forward tunnel in string format; maximum of 16 characters
`*rev_tnl_name`	The name of the reverse tunnel in string format; maximum of 16 characters

Output Parameters

None

Return Values

NSM_API_SET_SUCCESS when the function executed properly

NSM_API_SET_ERR_INVALID_FWD_TNL_NAME when the forward tunnel name is invalid

NSM_API_SET_ERR_FWD_TNL_ALREADY_ASSOCD when the forward tunnel is already associated with another tunnel

NSM_API_SET_ERR_FWD_TNL_DOES_NOT_EXIST when the forward tunnel does not exist

NSM_API_SET_ERR_FWD_TNL_NOT_IN_ASSOC_MODE when the forward tunnel is not in associated mode

NSM_API_SET_ERR_INVALID_ASSOCIATION when only ingress to egress tunnels or transit to tunnels can be associated

NSM_API_SET_ERR_INVALID_REV_TNL_NAME when the name for the reverse tunnel is invalid

NSM_API_SET_ERR_INVALID_ROLE_FOR_FWD_TNL when the node is neither Ingress nor Egress for the forward tunnel

NSM_API_SET_ERR_LSP_NOT_CONFD_FOR_FWD_TNL when the LSP is not configured for forward tunnel

NSM_API_SET_ERR_LSP_CONF_INVALID_FOR_FWD_TNL when the LSP configuration is invalid for the forward tunnel

NSM_API_SET_ERR_LSP_NOT_CONFD_FOR_REV_TNL when the LSP is not configured for reverse tunnel

NSM_API_SET_ERR_LSP_CONF_INVALID_FOR_REV_TNL when the MPLS-TP ME name exceeds maximum length

NSM_API_SET_ERR_REV_TNL_DOES_NOT_EXIST when the reverse tunnel does not exist

NSM_API_SET_ERR_REV_TNL_NOT_IN_ASSOC_MODE when the reverse tunnel is not in associated mode

NSM_API_SET_ERR_TNL_ASSOC_FAILURE when there is a tunnel association failure

NSM_API_SET_ERR_TNLS_OPERATE_ON_DIFF_END_POINTS when the forward and reverse tunnels do not operate between the same end points

nsm_api_mpls_tp_tnl_assoc_unconfig_validate

This function dissociates a forward tunnel with a reverse tunnel.

This function is called by the `no mpls-tp associate fwd-tunnel rev-tunnel` command.

Syntax

```
int
nsm_api_mpls_tp_tnl_assoc_unconfig_validate (u_int32_t vr_id,
                                             char *fwd_tnl_name,
                                             char *rev_tnl_name);
```

Input Parameters

`vr_id`	The virtual router (VR) instance ID
`*fwd_tnl_name`	The name of the forward tunnel in string format; maximum of 16 characters
`*rev_tnl_name`	The name of the reverse tunnel in string format; maximum of 16 characters

 © 2015 IP Infusion Inc. Proprietary

Output Parameters

None

Return Values

NSM_API_SET_SUCCESS when the function executed properly

NSM_API_SET_ERR_FWD_TNL_ALREADY_ASSOCD when the forward tunnel is already associated with another tunnel

NSM_API_SET_ERR_FWD_TNL_DOES_NOT_EXIST when the forward tunnel does not exist

NSM_API_SET_ERR_FWD_TNL_NOT_IN_ASSOC_MODE when the forward tunnel is not in associated mode

NSM_API_SET_ERR_INVALID_ASSOCIATION when the only ingress to egress tunnels or transit to transit tunnels can be associated

NSM_API_SET_ERR_INVALID_FWD_TNL_NAME when the forward tunnel name is invalid

NSM_API_SET_ERR_INVALID_REV_TNL_NAME when the name for the reverse tunnel is invalid

NSM_API_SET_ERR_INVALID_ROLE_FOR_FWD_TNL when the node is neither Ingress nor Egress for the specified forward tunnel

NSM_API_SET_ERR_LSP_CONF_INVALID_FOR_FWD_TNL when the LSP configuration is invalid for the forward tunnel

NSM_API_SET_ERR_LSP_NOT_CONFD_FOR_FWD_TNL when the LSP is not configured for forward tunnel

NSM_API_SET_ERR_LSP_NOT_CONFD_FOR_REV_TNL when the LSP is not configured for reverse tunnel

NSM_API_SET_ERR_REV_TNL_DOES_NOT_EXIST when the reverse tunnel does not exist

NSM_API_SET_ERR_REV_TNL_NOT_IN_ASSOC_MODE when the reverse tunnel not in associated mode

NSM_API_SET_ERR_REV_TNL_ALREADY_ASSOCD when the reverse tunnel is already associated with another tunnel

NSM_API_SET_ERR_TNLS_OPERATE_ON_DIFF_END_POINTS when the forward and reverse tunnels do not operate between the same end points

NSM_API_UNSET_ERR_TNL_DISSOC_FAILURE when the tunnels are disassociated

NSM_API_UNSET_ERR_TNLS_NOT_ASSOC_PREVIOUSLY when the tunnels were not previously set as associated

nsm_api_mpls_tp_tnl_mode_config

This function updates the tunnel mode.

This function is called by the `tunnel-mode` command.

Syntax

```
int
nsm_api_mpls_tp_tnl_mode_config (u_int32_t vr_id, u_char tp_tnl_mode,
                                 struct mpls_tp_tnl *tp_tnl);
```

Input Parameters

`vr_id`	The virtual router (VR) instance ID
`tnl_mode`	The MPLS-TP Tunnel mode: bidirectional; associated; unidirectional
`*tp_tnl`	The pointer to the MPLS-TP tunnel structure

Output Parameters

None

Return Values

NSM_API_SET_SUCCESS when the function executed properly

NSM_API_SET_ERR_MPLS_TP_LSP_CONFIG_EXIST when the MPLS-TP LSP configuration was previously applied for this tunnel

NSM_API_SET_ERR_MPLS_TP_TNL_MODE_INVALID when the MPLS-TP tunnel mode is invalid

NSM_API_SET_ERR_VR_NOT_EXIST when the virtual router does not exist

nsm_api_mpls_tp_lsp_config

This function validates and updates the LSP configuration for the tunnel. If the LSP is already configured, resetting the LSP ID is not allowed.

This function is called by the `lsp primary` and `bidirectional-lsp primary` commands

Syntax

```
int
nsm_api_mpls_tp_lsp_config (u_int32_t vr_id, u_int16_t lsp_id,
                            u_char lsp_type, u_char dir,
                            struct mpls_tp_tnl *tp_tnl,
                            struct mpls_tp_lsp **ret_lsp);
```

Input Parameters

`vr_id`	The virtual router (VR) instance ID
`lsp_id`	The LSP ID
`lsp_type`	The LSP type; primary or backup
`dir`	The LSP direction; unidirectional or bidirectional
`*tp_tnl`	The pointer to the MPLS-TP tunnel structure
`**ret_lsp`	The pointer to the MPLS-TP LSP structure pointer (double pointer)

Output Parameters

`**ret_lsp`	The pointer to the MPLS-TP LSP structure pointer (double pointer)

Return Values

NSM_API_SET_SUCCESS when the function executed properly

NSM_API_SET_ERROR when an invalid value was entered

NSM_API_SET_ERR_COROUTED_UNI_DIR_LSP_INVALID when a co-routed tunnel cannot be configured or un-configured as unidirectional LSP

NSM_API_SET_ERR_MPLS_TP_INVALID_LSP_TYPE when the MPLS-TP LSP type is invalid

NSM_API_SET_ERR_MPLS_TP_INVALID_LSP_DIR when the MPLS-TP LSP direction is invalid; only unidirectional or bidirectional are allowed

NSM_API_SET_ERR_MPLS_TP_LSP_ID_UPDATE_NOT_ALLOWED when the NHLFE or ILM entry exists; updating MPLS-TP LSP ID not allowed

 © 2015 IP Infusion Inc. Proprietary

NSM_API_SET_ERR_MPLS_TP_TNL_MODE_NOT_CONFIGURED

NSM_API_SET_ERR_NOT_COROUTED_BI_DIR_LSP_INVALID when an associated or bidirectional tunnel cannot be configured or un-configured as a bidirectional LSP

NSM_API_SET_ERR_VR_NOT_EXIST when the specified virtual router does not exist

nsm_api_mpls_tp_lsp_unconfig

This function removes the specified LSP configuration from the tunnel.

This function is called by the `no lsp primary` and `no bidirectional-lsp primary` commands

Syntax

```
int
nsm_api_mpls_tp_lsp_unconfig (u_int32_t vr_id, u_int16_t lsp_id,
                              u_char lsp_type, u_char dir,
                              struct mpls_tp_tnl *tp_tnl);
```

Input Parameters

`vr_id`	The virtual router (VR) instance ID
`lsp_id`	The LSP ID
`lsp_type`	The LSP type; primary or backup
`dir`	The LSP direction; unidirectional or bidirectional
`*tp_tnl`	The pointer to the MPLS-TP tunnel structure

Output Parameters

None

Return Values

NSM_API_SET_SUCCESS when the function executed properly

NSM_API_SET_ERR_COROUTED_UNI_DIR_LSP_INVALID when a co-routed tunnel cannot be configured or un-configured as unidirectional LSP

NSM_API_SET_ERR_MPLS_TP_INVALID_LSP_DIR when the MPLS-TP LSP direction is invalid; only unidirectional or bidirectional are allowed

NSM_API_SET_ERR_MPLS_TP_INVALID_LSP_TYPE when the MPLS-TP LSP type is invalid

NSM_API_SET_ERR_MPLS_TP_LSP_DOES_NOT_EXIST when the MPLS-TP LSP does not exist

NSM_API_SET_ERR_MPLS_TP_TNL_ASSOC_EXIST when the MPLS-TP tunnel is associated with another tunnel

NSM_API_SET_ERR_MPLS_TP_TNL_HAS_MEG_CONFIG when the MPLS-TP tunnel has a MEG association

NSM_API_SET_ERR_NOT_COROUTED_BI_DIR_LSP_INVALID when an associated or bidirectional tunnel cannot be configured or un-configured as a bidirectional LSP

NSM_API_SET_ERR_VR_NOT_EXIST when the virtual router does not exist

nsm_api_mpls_tp_nhlfe_config

This function validates and creates/updates NHLFE entry for a Tunnel's LSP at the end point(s).

This function processes the `nhlfe-entry` command.

Syntax

```
int
nsm_api_mplstp_nhlfe_config (u_int32_t vr_id,
                             struct mpls_tp_lsp *lsp, u_char dir,
                             u_int32_t ifindex, u_int32_t label,
                             struct nsm_mpls_bw_class *bw_class,
                             char *mac_str);
```

Input Parameters

`vr_id`	The virtual router (VR) instance ID
`*lsp`	The pointer to MPLS-TP LSP structure
`dir`	Designates NHLFE for the LSP: Forward or Reverse
`ifindex`	The outgoing interface index
`label`	The outgoing label
`*bw_class`	The pointer to BW Class structure; this input parameter is optional
`mac_str`	The MAC address of the nexthop; this input parameter is optional

Output Parameters

None

Return Values

NSM_API_SET_SUCCESS when the function executed properly

NSM_API_SET_ERR_VR_NOT_EXIST when the virtual router does not exist

NSM_API_SET_ERR_MPLS_TP_ASSOC_TNL_UPDATE when the tunnel is associated; this update is not supported

NSM_API_SET_ERR_MPLS_TP_COROUTED_INTF_CHECK when the forward and reverse paths of a co-routed tunnel must be defined on the same interface

NSM_API_SET_ERR_MPLS_TP_INVALID_OUT_LABEL when the outgoing label is invalid

NSM_API_SET_ERR_MPLS_TP_MAC_ADDR_INVALID when the MAC address format is invalid

NSM_API_SET_ERR_MPLS_TP_MAC_ADDR_RESERVED when the reserved MAC address is not allowed

NSM_API_SET_ERR_MPLS_TP_MAC_ADDR_TRANSLATE when unable to translate the specified MAC address

NSM_API_SET_ERR_MPLS_TP_MAC_UPDATE when updating the nexthop MAC is not supported

NSM_API_SET_ERR_MPLS_TP_NHLFE_CMD_FAILURE when NHLFE command failure

NSM_API_SET_ERR_MPLS_TP_OUT_DL_ID_NOT_SET when the local-interface-id is not set for outgoing provider interface

NSM_API_SET_ERR_MPLS_TP_OUT_IF_LOOPBACK when the outgoing MPLS-TP provider interface cannot be set as loopback

NSM_API_SET_ERR_MPLS_TP_OUT_IF_NO_DL when the MPLS-TP data link does not exist on the outgoing provider interface

NSM_API_SET_ERR_MPLS_TP_OUT_IF_NON_GMPLS_DC when the outgoing MPLS-TP provider interface is not operating in GMPLS data-control mode

NSM_API_SET_ERR_MPLS_TP_OUT_LABEL_UPDATE when updating the outgoing label is not supported

 © 2015 IP Infusion Inc. Proprietary

NSM_API_SET_ERR_MPLS_TP_OUT_INTF_UPDATE when updating the outgoing interface is not supported

NSM_API_SET_ERR_MPLS_TP_UPDATE_NOT_SUPPORTED when this update is not supported

nsm_api_mpls_tp_nhlfe_unconfig

This function validates and then deletes NHLFE entry for a Tunnel's LSP at the end point(s).

This function processes the `no nhlfe-entry` command.

Syntax

```
    int
nsm_api_mpls_tp_nhlfe_unconfig (u_int32_t vr_id,
                                struct mpls_tp_lsp *lsp, u_char dir,
                                u_int32_t ifindex, u_int32_t label);
```

Input Parameters

`vr_id`	The virtual router (VR) instance ID
`*lsp`	The pointer to MPLS-TP LSP structure
`dir`	Indicates NHLFE for the LSP: Forward or Reverse
`ifindex`	The outgoing interface index
`label`	The outgoing label

Output Parameters

None

Return Values

NSM_API_SET_SUCCESS when the function executed properly

NSM_API_SET_ERR_MPLS_TP_NHLFE_NOT_EXISTS when the NHLFE entry does not exist

NSM_API_SET_ERR_MPLS_TP_INVALID_OUT_LABEL when the outgoing label is invalid

NSM_API_SET_ERR_MPLS_TP_OUT_IF_LOOPBACK when the outgoing MPLS-TP provider interface cannot be set as loopback

NSM_API_SET_ERR_MPLS_TP_OUT_IF_NON_GMPLS_DC when the outgoing MPLS-TP provider interface is not operating in GMPLS data-control mode

NSM_API_SET_ERR_MPLS_TP_OUT_IF_NO_DL when the MPLS-TP datalink does not exist on the outgoing provider interface

NSM_API_SET_ERR_MPLS_TP_OUT_DL_ID_NOT_SET when the local-interface-id is not set for outgoing provider interface

NSM_API_SET_ERR_MPLS_TP_ASSOC_NHLFE_DEL when the NHLFE is bound to another tunnel; the NHLFE cannot be unconfigured

NSM_API_SET_ERR_MPLS_TP_NHLFE_NOT_EXISTS when the NHLFE entry does not exist

nsm_api_mpls_tp_ilm_swap_config

This function validates and creates/updates the specified ILM entry for a Tunnel's LSP on Transit nodes.

This function processes the `ilm-entry swap` command.

Syntax

```
int
nsm_api_mpls_tp_ilm_swap_config (u_int32_t vr_id, struct mpls_tp_lsp *lsp,
                                 u_char dir, u_int32_t in_ifindex,
                                 u_int32_t in_label, u_int32_t out_ifindex,
                                 u_int32_t out_label,
                                 struct nsm_mpls_bw_class *bw_class,
                                 char *mac_str);
```

Input Parameters

`vr_id`	The virtual router (VR) instance ID
`*lsp`	Pointer to MPLS-TP LSP structure
`dir`	Identifies NHLFE for the LSP: Forward or Reverse
`in_ifindex`	The incoming interface index
`in_label`	The incoming label
`out_ifindex`	The outgoing interface index
`out_label`	The outgoing label
`*bw_class`	The pointer to BW Class structure: this input parameter is optional
`*mac_str`	The MAC address of the nexthop; this input parameter is optional

Output Parameters

None

Return Values

NSM_API_SET_SUCCESS when the function executed properly

NSM_API_SET_ERR_VR_NOT_EXIST when the virtual router does not exist

NSM_API_SET_ERR_MPLS_TP_INVALID_OUT_LABEL when the outgoing label is invalid

NSM_API_SET_ERR_MPLS_TP_INVALID_IN_LABEL when the incoming label is invalid

NSM_API_SET_ERR_MPLS_TP_OUT_IF_LOOPBACK when the outgoing MPLS-TP provider interface cannot be set as loopback

NSM_API_SET_ERR_MPLS_TP_IN_IF_LOOPBACK when the incoming MPLS-TP provider interface cannot be set as a loopback

NSM_API_SET_ERR_MPLS_TP_OUT_IF_NON_GMPLS_DC when the outgoing MPLS-TP provider interface is not operating in GMPLS data-control mode

NSM_API_SET_ERR_MPLS_TP_IN_IF_NON_GMPLS_DC when the incoming MPLS-TP provider interface is not operating in GMPLS data-control mode

NSM_API_SET_ERR_MPLS_TP_OUT_IF_NO_DL when the MPLS-TP datalink does not exist on the outgoing provider interface

NSM_API_SET_ERR_MPLS_TP_IN_IF_NO_DL when the MPLS-TP datalink does not exist on the incoming provider interface

NSM_API_SET_ERR_MPLS_TP_OUT_DL_ID_NOT_SET when the local-interface-id is not set for outgoing provider interface

© 2015 IP Infusion Inc. Proprietary

NSM_API_SET_ERR_MPLS_TP_IN_DL_ID_NOT_SET when the local-interface-id not set as an incoming provider interface

NSM_API_SET_ERR_MPLS_TP_ILM_SAME_INTF when incoming and outgoing interfaces can not be the same

NSM_API_SET_ERR_MPLS_TP_MAC_ADDR_TRANSLATE when unable to translate the specified MAC address

NSM_API_SET_ERR_MPLS_TP_MAC_ADDR_RESERVED when the reserved MAC address is not allowed

NSM_API_SET_ERR_MPLS_TP_MAC_ADDR_INVALID when the MAC address format is invalid

NSM_API_SET_ERR_MPLS_TP_COROUTED_INTF_CHECK when the forward and reverse paths of a co-routed tunnel must be defined on the same interface

NSM_API_SET_ERR_MPLS_TP_ASSOC_TNL_UPDATE when the tunnel is associated; this updated is not supported

NSM_API_SET_ERR_MPLS_TP_UPDATE_NOT_SUPPORTED when this update is not supported

NSM_API_SET_ERR_MPLS_TP_OUT_LABEL_UPDATE when updating the outgoing label is not supported

NSM_API_SET_ERR_MPLS_TP_IN_LABEL_UPDATE when updating the incoming label not supported

NSM_API_SET_ERR_MPLS_TP_OUT_INTF_UPDATE when updating the outgoing interface is not supported

NSM_API_SET_ERR_MPLS_TP_IN_INTF_UPDATE when updating the incoming interface not supported

NSM_API_SET_ERR_MPLS_TP_MAC_UPDATE when updating the nexthop MAC is not supported

NSM_API_SET_ERR_MPLS_TP_DUPLICATE_ILM when the label is already in use; enter another label

NSM_API_SET_ERR_MPLS_TP_ILM_CMD_FAILURE when the ILM command failed

nsm_api_mpls_tp_ilm_swap_unconfig

This function validates and deletes the specified ILM entry for a Tunnel's LSP on Transit nodes.

This function is called by the `no ilm-entry swap` command.

Note: The command `no ilm-entry LABEL IFNAME swap LABEL IFNAME` calls the internal command `nsm_mplstp_ilm_swap_unconfig`, which then calls the function `nsm_api_mpls_tp_ilm_swap_unconfig`. Internal commands are outside the scope of this document; refer to the source file `nsm/mplstp/nsm_mplstp_cli.c`.

Syntax

```
int
nsm_api_mpls_tp_ilm_swap_unconfig (u_int32_t vr_id,
                                   struct mpls_tp_lsp *lsp, u_char dir,
                                   u_int32_t in_ifindex, u_int32_t in_label,
                                   u_int32_t out_ifindex,u_int32_t out_label);
```

Input Parameters

`vr_id`	The virtual router (VR) instance ID
`*lsp`	Pointer to MPLS-TP LSP structure
`dir`	Identifies NHLFE for the LSP: Forward or Reverse
`in_ifindex`	The incoming interface index
`in_label`	The incoming label

Output Parameters

None

Return Values

NSM_API_SET_SUCCESS when the function executed properly

NSM_API_SET_ERR_VR_NOT_EXIST when the virtual router does not exist

NSM_API_SET_ERR_MPLS_TP_ILM_NOT_EXISTS when the ILM entry does not exist

NSM_API_SET_ERR_MPLS_TP_INVALID_OUT_LABEL when the outgoing label is invalid

NSM_API_SET_ERR_MPLS_TP_INVALID_IN_LABEL when the incoming label is invalid

NSM_API_SET_ERR_MPLS_TP_OUT_IF_LOOPBACK when the outgoing MPLS-TP provider interface cannot be set as loopback

NSM_API_SET_ERR_MPLS_TP_IN_IF_LOOPBACK when the incoming MPLS-TP provider interface cannot be set as a loopback

NSM_API_SET_ERR_MPLS_TP_OUT_IF_NON_GMPLS_DC when the outgoing MPLS-TP provider interface is not operating in GMPLS data-control mode

NSM_API_SET_ERR_MPLS_TP_IN_IF_NON_GMPLS_DC when the incoming MPLS-TP provider interface is not operating in GMPLS data-control mode

NSM_API_SET_ERR_MPLS_TP_OUT_IF_NO_DL when the MPLS-TP datalink does not exist on the outgoing provider interface

NSM_API_SET_ERR_MPLS_TP_IN_IF_NO_DL when the MPLS-TP datalink does not exist on the incoming provider interface

NSM_API_SET_ERR_MPLS_TP_OUT_DL_ID_NOT_SET when the local-interface-id is not set for outgoing provider interface

NSM_API_SET_ERR_MPLS_TP_IN_DL_ID_NOT_SET when the local-interface-id not set as an incoming provider interface

NSM_API_SET_ERR_MPLS_TP_ASSOC_ILM_DEL when bound to another tunnel; ILM cannot be unconfigured

nsm_api_mpls_tp_ilm_pop_config

This function validates and creates/updates ILM entry for a Tunnel's LSP on end point(s).

This function processes the `ilm-entry pop` command.

Syntax

```
int
nsm_api_mpls_tp_ilm_pop_config(u_int32_t vr_id, struct mpls_tp_lsp *lsp,
                               u_char dir, u_int32_t in_ifindex,
                               u_int32_t in_label);
```

Input Parameters

`vr_id`	The virtual router (VR) instance ID
`*lsp`	Pointer to MPLS-TP LSP structure
`dir`	Identifies NHLFE for the LSP: Forward or Reverse
`in_ifindex`	The incoming interface index

 © 2015 IP Infusion Inc. Proprietary

`in_label`	The incoming label

Output Parameters

None

Return Values

NSM_API_SET_SUCCESS when the function executed properly

NSM_API_SET_ERR_VR_NOT_EXIST when the virtual router does not exist

NSM_API_SET_ERR_MPLS_TP_INVALID_IN_LABEL when the incoming label is invalid

NSM_API_SET_ERR_MPLS_TP_IN_IF_LOOPBACK when the incoming MPLS-TP provider interface cannot be set as a loopback

NSM_API_SET_ERR_MPLS_TP_IN_IF_NON_GMPLS_DC when the incoming MPLS-TP provider interface is not operating in GMPLS data-control mode

NSM_API_SET_ERR_MPLS_TP_IN_IF_NO_DL when the MPLS-TP datalink does not exist on the incoming provider interface

NSM_API_SET_ERR_MPLS_TP_IN_DL_ID_NOT_SET when the local-interface-id not set as an incoming provider interface

NSM_API_SET_ERR_MPLS_TP_COROUTED_INTF_CHECK when the forward and reverse paths of a co-routed tunnel must be defined on the same interface

NSM_API_SET_ERR_MPLS_TP_ASSOC_TNL_UPDATE when the tunnel is associated; this updated is not supported

NSM_API_SET_ERR_MPLS_TP_UPDATE_NOT_SUPPORTED when this update is not supported

NSM_API_SET_ERR_MPLS_TP_IN_LABEL_UPDATE when updating the incoming label not supported

NSM_API_SET_ERR_MPLS_TP_IN_INTF_UPDATE when updating the incoming interface not supported

NSM_API_SET_ERR_MPLS_TP_DUPLICATE_ILM when the label is already in use; enter another label

NSM_API_SET_ERR_MPLS_TP_ILM_CMD_FAILURE when the ILM command failed

nsm_api_mpls_tp_ilm_pop_unconfig

This function validates and deletes the ILM entry for a Tunnel's LSP end point(s).

This function processes the `no ilm-entry` command

Syntax

```
int
nsm_api_mpls_tp_ilm_pop_unconfig (u_int32_t vr_id, struct mpls_tp_lsp *lsp,
                                  u_char dir, u_int32_t in_ifindex,
                                  u_int32_t in_label);
```

Input Parameters

`vr_id`	The virtual router (VR) instance ID
`*lsp`	Pointer to MPLS-TP LSP structure
`dir`	Identifies NHLFE for the LSP: Forward or Reverse
`in_ifindex`	The incoming interface index
`in_label`	The incoming label

Output Parameters

None

Return Values

NSM_API_SET_SUCCESS when the function executed properly

NSM_API_SET_ERR_VR_NOT_EXIST when the virtual router does not exist

NSM_API_SET_ERR_MPLS_TP_ILM_NOT_EXISTS when the ILM entry does not exist

NSM_API_SET_ERR_MPLS_TP_INVALID_IN_LABEL when the incoming label is invalid

NSM_API_SET_ERR_MPLS_TP_IN_IF_LOOPBACK when the incoming MPLS-TP provider interface cannot be set as a loopback

NSM_API_SET_ERR_MPLS_TP_IN_IF_NON_GMPLS_DC when the incoming MPLS-TP provider interface is not operating in GMPLS data-control mode

NSM_API_SET_ERR_MPLS_TP_IN_IF_NO_DL when the MPLS-TP datalink does not exist on the incoming provider interface

NSM_API_SET_ERR_MPLS_TP_IN_DL_ID_NOT_SET when the local-interface-id not set as an incoming provider interface

NSM_API_SET_ERR_MPLS_TP_ASSOC_ILM_DEL when bound to another tunnel; ILM cannot be unconfigured

NSM_API_SET_ERR_MPLS_TP_ILM_CMD_FAILURE when the ILM command failed

nsm_mpls_oam_send_vc_update

This function sends a VC update message from NSM to OAMD. This message (IPC) is sent whenever a pseudowire (PW) CLI is added, updated, or deleted. In addition, it is sent when the PW is installed into the FIB to update interface related information.

Syntax

```
void
nsm_mpls_oam_send_vc_update (struct nsm_master *nm, struct nsm_mpls_circuit *vc)
```

Input Parameters

`*nm`	NSM master

Output Parameters

`*vc`	VC update message

Return Values

None

nsm_mpls_l2_circuit_send_pw_status

This function sends the pseudowire status from NSM to OAMD. Whenever there is any change in the PW status NSM triggers this message (IPC).

Syntax

```
void
```

 © 2015 IP Infusion Inc. Proprietary

```
nsm_mpls_l2_circuit_send_pw_status (struct nsm_mpls_if *mif,
                                    struct nsm_mpls_circuit *vc)
```

Input Parameters

*mif	MPLS interface

Output Parameters

*vc	VC update message

Return Values

None

mpls_l2_circuit_create_cli

The API creates a layer 2 circuit.

Syntax

```
int
mpls_l2_circuit_create_cli (struct cli *cli, char *vc_name, char *val_str,
                            char *peer_gid_str, char *peer_ac_id_str,
                            char *address, char *group_name, u_int32_t group_id,
                            u_char c_word, char mapping_type,
                            char *tunnel_info, u_char tunnel_dir,
                            u_int8_t cc_types, u_int8_t cv_types,
                            u_int8_t fec_type_vc, u_int8_t is_passive)
```

Input Parameters

*cli	Specifies the instance of the CLI structure.
*vc_name	Port used for a VC name
*val_str	Value string
*peer_gid_str	Peer Global ID string
*peer_ac_id_str	Peer attachment circuit ID string
*address	Address
*group_name	Group name
group_id	Group ID
c_word	Control word
mapping_type	Mapping type
*tunnel_info	Tunnel information
tunnel_dir	Tunnel type
cc_types	Control Channel Types
cv_types	Control Verification Types
fec_type_vc	FEC VC type
is_passive	Is passive

Output Parameters

None

Return Values

CLI_ERROR

CLI_SUCCESS

nsm_mpls_l2_circuit_update

The API updates a layer 2 circuit.

Syntax

```
static int
nsm_mpls_l2_circuit_update (struct nsm_master *nm, char *vc_name,
                            u_int32_t vc_id, struct pal_in4_addr *egress,
                            u_int32_t peer_gid, u_int32_t peer_ac_id,
                            char *group_name, u_int32_t group_id, u_char c_word,
                            u_char mapping_type, void *tunnel_info,
                            u_char tunnel_dir, u_int8_t fec_type_vc,
                            u_int8_t cc_types, u_int8_t cv_types,
                            struct nsm_mpls_circuit *vc)
```

Input Parameters

*nm	NSM master
*vc_name	Port used for a VC name
vc_id	VC ID
*egress	Egress type
*peer_gid	Peer Global ID
*peer_ac_id	Peer attachment circuit ID
*group_name	Group name
group_id	Group ID
c_word	Control word
mapping_type	Mapping type
*tunnel_info	Tunnel information
tunnel_dir	Tunnel type
fec_type_vc	FEC VC type
cc_types	Control Channel Types
cv_types	Control Verification Types
vc	Virtual circuit

Output Parameters

None

© 2015 IP Infusion Inc. Proprietary

Return Values

CLI_ERROR

CLI_SUCCESS

nsm_server_recv_pw_status

This function processes the remote PW status received from a protocol module, such as from LDPD or OAMD.

Syntax

```
s_int32_t
nsm_server_recv_pw_status (struct nsm_msg_header *header,
                           void *arg, void *message)
```

Input Parameters

`*header`	Header type
`*arg`	Argument type
`*message`	Message type

Output Parameters

None

Return Values

`None`

nsm_api_itut_mpls_tp_ring_config

This function creates a new MPLS-TP ring tunnel, if not already created.

Syntax

```
s_int32_t nsm_api_itut_mpls_tp_ring_config (u_int32_t vr_id,
                                            u_int32_t ring_id,struct mpls_tp_tnl **tnl)
```

Input Parameters

`vr_id`	VR instance ID
`ring_id`	**Ring tunnel ID**

Output Parameters

`tnl`	Pointer to the MPLS-TP tunnel structure

Return Values

NSM_API_SET_SUCCESS when the function succeeds

NSM_API_SET_ERROR when there is an internal error

NSM_API_SET_ERR_VR_NOT_EXIST when the VR does not exist

NSM_API_SET_ERR_MPLS_TP_ID_CONFIG_NOT_EXISTS when a ring tunnel with the given ID does not exist.

NSM_API_SET_ERR_MPLS_TP_RING_DUPLICATE_TNL_ID when a tunnel with the given ID already exists.

nsm_api_itut_mpls_tp_ring_unconfig

This function destroys an MPLS-TP ring tunnel, if already created.

Syntax

```
s_int32_t nsm_api_itut_mpls_tp_ring_unconfig (u_int32_t vr_id,
                                              u_int32_t ring_id)
```

Input Parameters

vr_id	VR instance ID
ring_id	**Ring tunnel ID**

Output Parameters

None

Return Values

NSM_API_SET_SUCCESS when the function succeeds

NSM_API_SET_ERROR when there is an internal error

NSM_API_SET_ERR_VR_NOT_EXISTwhen the VR does not exist

NSM_API_SET_ERR_MPLS_TP_RING_DOES_NOT_EXIST when a ring tunnel with the given ID does not exits

NSM_API_SET_ERR_MPLS_TP_RING_HAS_MEG_CONFIG when a ring tunnel with the given ID does not exits

oam_pw_recv_nsm_pw_status

This function processes the local pseudowire (PW) status message from NSM. If there is a change in the PW status, it will form the PW OAM message and send the PW status TLV to the other PW endpoint.

Syntax

```
void
oam_pw_recv_nsm_pw_status (struct bfd_master *bm,
                           struct nsm_msg_pw_status *msg,
                           struct oam_vc_cache *vc_cache)
```

Input Parameters

*bm	Pointer to the BFD master
*msg	Message type
*vc_cache	Virtual circuit cache.

Output Parameters

None

Return Values

0

© 2015 IP Infusion Inc. Proprietary

oam_pw_status_create_send

This function creates an OAM message for local PW status and then sends it to a remote peer.

Syntax

```
int
oam_pw_status_create_send (struct bfd_master *bm,
                           struct nsm_msg_pw_status *msg,
                           struct oam_vc_cache *vc_cache,
                           bool_t skip_gal)
```

Input Parameters

*bm	Pointer to the BFD master
*msg	Message type
*vc_cache	Virtual circuit cache.
skip_gal	

Output Parameters

None

Return Values

```
0
MPLS_OAM_ERR_EXPLICIT_NULL
```

oam_pw_msg_packet_read

This function reads an OAM message from the forwarder.

Syntax

```
s_int8_t
oam_pw_msg_packet_read (struct bfd_master *lbm, u_char *buf, u_int16_t len,
                        struct mpls_oam_ctrl_data *ctrl_data)
```

Input Parameters

*lbm	Pointer to the LBM master
*msg	Message type
*vc_cache	Virtual circuit cache.
skip_gal	Skip gal

Output Parameters

None

Return Values

```
0
MPLS_OAM_ERR_EXPLICIT_NULL
```

oam_pw_process_ack_msg

This function copies a refresh timer value and processes the ACK message of remote peer.

Syntax

```
s_int8_t
oam_pw_process_ack_msg (struct oam_vc_cache *vc_cache,
                        struct oam_pw_msg *msg)
```

Input Parameters

*vc_cache	Virtual circuit cache.
*msg	Message type

Output Parameters

None

Return Values

0

oam_pw_process_status_msg

This function processes the remote peer new status message and sends reply message.

Syntax

```
s_int8_t
oam_pw_process_status_msg (struct bfd_master *bm,
                           struct oam_vc_cache *vc_cache,
                           struct oam_pw_msg *msg)
```

Input Parameters

*bm	Pointer to the BFD master
*vc_cache	Virtual circuit cache.
*msg	Message type

Output Parameters

None

Return Values

0

© 2015 IP Infusion Inc. Proprietary

oam_mpls_process_oam_update

This function processes the VC update message from NSM. It creates the VC Cache in OAMD (if not already present) and updates the parameters of the VC received into its database.

Syntax

```
int
oam_mpls_process_oam_update (struct bfd_master *bm,
                             struct nsm_msg_oam_update *msg)
```

Input Parameters

*bm	Pointer to the BFD master
*msg	Message type

Output Parameters

None

Return Values

0

oam_pw_send_status_timer

This function processes the VC update message from NSM. It creates the VC Cache in OAMD (if not already present) and updates the parameters of the VC received into its database.

Syntax

```
int
oam_mpls_process_oam_update (struct bfd_master *bm,
                             struct nsm_msg_oam_update *msg)
```

Input Parameters

*bm	Pointer to the BFD master
*msg	Message type

Output Parameters

None

Return Values

0

© 2015 IP Infusion Inc. Proprietary

CHAPTER 20 MPLS-TP OAM Command API

This chapter describes the functions that support Operation Administration Maintenance (OAM) configurations.

oam_mplstp_api_meg_create

This function creates a MPLS-TP Maintenance Entity Group (MEG) if it is not already present. If the MEG is already configured, the existing MEG entry structure associated with it is returned.

This function is called by the `ietf meg` command.

Syntax

```
s_int32_t
oam_mplstp_api_meg_create (u_int32_t vr_id, char *meg_name);
```

Input Parameters

`vr_id`	The virtual router (VR) instance ID
`meg_name`	Character pointer to the MEG name

Output Parameters

None

Return Values

MPLSTP_API_SUCCESS when the function succeeds

MPLSTP_ERR_VR_NOT_EXISTS when the virtual router cannot be found

MPLSTP_ERR_MEG_NAME_INVALID when `meg_name` is NULL

MPLSTP_ERR_MEG_NAME_EXCEEDS_MAX_LEN when the MEG name exceeds the maximum length

MPLSTP_ERR_MEG_CREATE_FAILED when MEG creation fails

MPLSTP_API_FAILURE when the function fails

oam_mplstp_api_meg_delete

This function deletes the MPLS-TP Maintenance Entity Group. If the MEG is not already created, an error is returned.

This function is called by the `no ietf meg` command.

```
Syntax
s_int32_t
oam_mplstp_api_meg_delete (u_int32_t vr_id, char *meg_name);
```

Input Parameters

	The virtual router (VR) instance ID
`meg_name`	Character pointer containing the MEG name

Output Parameters

None

Return Values

MPLSTP_API_SUCCESS when the function succeeds

MPLSTP_ERR_VR_NOT_EXISTS when the virtual router cannot be found

MPLSTP_ERR_MEG_NAME_INVALID when `meg_name` is NULL

MPLSTP_ERR_MEG_NAME_EXCEEDS_MAX_LEN when the MEG name exceeds the maximum length

MPLSTP_ERR_MEG_ENTRY_NOT_EXISTS when the MEG entry cannot be found

MPLSTP_ERR_ME_ENTRY_EXISTS when the ME entry does not exist

oam_mplstp_api_set_service_type

This function sets the type of service of the MPLS-TP Maintenance Entity Group. No update is allowed if any ME exists.

Syntax

```
s_int32_t
oam_mplstp_api_set_service_type (u_int32_t vr_id, char *meg_name,
                                 u_int8_t service_type)
```

Input Parameters

`vr_id`	The virtual router (VR) instance ID
`meg_name`	Character pointer containing the MEG name
`service-type`	The MEG service type; either TUNNEL, or VC or DATALINK

Output Parameters

None

Return Values

MPLSTP_API_SUCCESS when the function executed properly

MPLSTP_ERR_VR_NOT_EXISTS when the virtual router cannot be found

MPLSTP_ERR_MEG_NAME_INVALID when `meg_name` is NULL

MPLSTP_ERR_MEG_NAME_EXCEEDS_MAX_LEN when the MEG name exceeds the maximum length

MPLSTP_ERR_MEG_ENTRY_NOT_EXISTS when the MEG entry cannot be found

MPLSTP_API_ERR_CANT_MODIFY_SERVICE_TYPE when an attempt is made to modify the service type

oam_mplstp_api_me_create

This function creates a Maintenance Entity (ME) in MPLS-TP Maintenance Entity Group (MEG)

Syntax

```
s_int32_t
oam_mplstp_api_me_create (u_int32_t vr_id, char *meg_name, char *me_name);
```


© 2015 IP Infusion Inc. Proprietary

Input Parameters

`vr_id`	VR instance ID
`meg_name`	Pointer to the MEG name.
`me_name`	The ME name

Output Parameters

None

Return Values

MPLSTP_API_SUCCESS when the function completes successfully

MPLSTP_ERR_VR_NOT_EXISTS when the virtual router cannot be found

MPLSTP_ERR_MEG_NAME_INVALID when `meg_name` is NULL

MPLSTP_ERR_ME_NAME_INVALID when the ME name is invalid

MPLSTP_ERR_MEG_NAME_EXCEEDS_MAX_LEN when the MEG name exceeds the maximum length

MPLSTP_ERR_ME_NAME_EXCEEDS_MAX_LEN when the ME name given exceeds the maximum length

MPLSTP_ERR_MEG_ENTRY_NOT_EXISTS when the MEG entry cannot be found

MPLSTP_ERR_ME_ENTRY_ALREADY_EXISTS when the maintenance entity already exists

MPLSTP_ERR_ME_CREATE_FAILED when the ME creation process fails

MPLSTP_API_FAILURE when the function fails

oam_mplstp_api_me_delete

This function deletes a Maintenance Entity in MPLS-TP Maintenance Entity Group.

Syntax

```
s_int32_t
oam_mplstp_api_me_delete (u_int32_t vr_id, char *meg_name, char *me_name);
```

Input Parameters

`vr_id`	The VR instance ID
`meg_name`	Pointer to the MEG name.
`me_name`	The ME name

Output Parameters

None

Return Values

MPLSTP_API_SUCCESS when the function completes successfully

MPLSTP_ERR_VR_NOT_EXISTS when the virtual router cannot be found

MPLSTP_ERR_MEG_NAME_INVALID when `meg_name` is NULL

MPLSTP_ERR_ME_NAME_INVALID when the ME name is invalid

MPLSTP_ERR_MEG_NAME_EXCEEDS_MAX_LEN when the MEG name given exceeds the maximum length

MPLSTP_ERR_ME_NAME_EXCEEDS_MAX_LEN when the ME name given exceeds the maximum length

MPLSTP_ERR_MEG_ENTRY_NOT_EXISTS when the MEG entry cannot be found

MPLSTP_ERR_ME_ENTRY_NOT_EXISTS when the ME entry cannot be found

MPLSTP_ERR_OAM_CONFIG_EXISTS when an OAM configuration exists

oam_mplstp_api_associate_datalink

This function associates an MPLS-TP provider interface to an MPLS-TP Maintenance Entity.

Note: Once the data link association is set, no update is allowed.

Syntax

```
s_int32_t
oam_mplstp_api_associate_datalink (u_int32_t vr_id, char *meg_name,
                                   char *me_name, char *if_name)
```

Input Parameters

`vr_id`	The VR instance ID
`meg_name`	Pointer to the MEG name.
`me_name`	The ME name
`if_name`	The interface name

Output Parameters

None

Return Values

MPLSTP_API_SUCCESS when the function processing is successful

MPLSTP_ERR_VR_NOT_EXISTS when the virtual router cannot be found

MPLSTP_ERR_MEG_NAME_INVALID when `meg_name` is NULL

MPLSTP_ERR_ME_NAME_INVALID when the ME name is invalid

MPLSTP_ERR_MEG_NAME_EXCEEDS_MAX_LEN when the MEG name exceeds the maximum length

MPLSTP_ERR_ME_NAME_EXCEEDS_MAX_LEN when the ME name exceeds the maximum length

MPLSTP_ERR_ME_IF_NAME_INVALID when the interface name given is invalid

MPLSTP_ERR_ME_IF_NAME_EXCEEDS_MAX_LEN when the interface name exceeds the maximum length

MPLSTP_ERR_MEG_ENTRY_NOT_EXISTS when the MEG entry cannot be found

MPLSTP_ERR_ME_SERVICE_TYPE_NOT_MATCH when the ME service type does not match

MPLSTP_ERR_ME_ENTRY_NOT_EXISTS when the ME entry cannot be found

MPLSTP_ERR_ME_CANT_UPDATE_SERVICE_ID when an attempt is made to update the association

oam_mplstp_api_associate_tnl

This function associates an MPLS-TP tunnel to an MPLS-TP Maintenance Entity when the service type configured is Tunnel.

 © 2015 IP Infusion Inc. Proprietary

Note: Once the tunnel association is set, no update is allowed.

Syntax

```
s_int32_t
oam_mplstp_api_associate_tnl (u_int32_t vr_id, char *meg_name, char *me_name,
                              char *tnl_name)
```

Input Parameters

`vr_id`	VR instance ID
`meg_name`	Pointer to the MEG name.
`me_name`	The ME name
`tnl_name`	The tunnel name

Output Parameters

None

Return Values

MPLSTP_API_SUCCESS when the function succeeds

MPLSTP_API_SUCCESS when the function processing is successful

MPLSTP_ERR_VR_NOT_EXISTS when the virtual router cannot be found

MPLSTP_ERR_MEG_NAME_INVALID when `meg_name` is NULL

MPLSTP_ERR_ME_NAME_INVALID when the ME name is invalid

MPLSTP_ERR_MEG_NAME_EXCEEDS_MAX_LEN when the MEG name exceeds the maximum length

MPLSTP_ERR_ME_NAME_EXCEEDS_MAX_LEN when the ME name exceeds the maximum length

MPLSTP_ERR_ME_TNL_NAME_INVALID when `tnl_name` is NULL

MPLSTP_ERR_ME_TNL_NAME_EXCEEDS_MAX_LEN when `tnl_name` exceeds the maximum length

MPLSTP_ERR_MEG_ENTRY_NOT_EXISTS when the MEG entry cannot be found

MPLSTP_ERR_ME_SERVICE_TYPE_NOT_MATCH when the ME service type does not match

MPLSTP_ERR_ME_ENTRY_NOT_EXISTS when the ME entry cannot be found

MPLSTP_ERR_ME_CANT_UPDATE_SERVICE_ID when an attempt is made to update the association

oam_mplstp_api_associate_vc

This function associates and MPLS-TP VC to an MPLS-TP Maintenance Entity, when the service type configured is tunnel.

Note: Once the VC association is set, no update is allowed.

Syntax

```
s_int32_t
oam_mplstp_api_associate_vc (u_int32_t vr_id, char *meg_name, char *me_name,
                             u_int32_t vcid)
```

Input Parameters

`vr_id`	VR instance ID
`meg_name`	Pointer to the MEG name.
`me_name`	The ME name
`vcid`	Virtual circuit ID

Output Parameters

None

Return Values

MPLSTP_API_SUCCESS when the function processing is successful

MPLSTP_ERR_VR_NOT_EXISTS when the virtual router cannot be found

MPLSTP_ERR_MEG_NAME_INVALID when `meg_name` is NULL

MPLSTP_ERR_ME_NAME_INVALID when the ME name is invalid

MPLSTP_ERR_MEG_NAME_EXCEEDS_MAX_LEN when the MEG name exceeds the maximum length

MPLSTP_ERR_ME_NAME_EXCEEDS_MAX_LEN when the ME name exceeds the maximum length

MPLSTP_ERR_ME_VC_ID_INVALID when the Virtual Circuit ID is invalid

MPLSTP_ERR_MEG_ENTRY_NOT_EXISTS when the MEG entry cannot be found

MPLSTP_ERR_ME_SERVICE_TYPE_NOT_MATCH when the ME service type does not match

MPLSTP_ERR_ME_ENTRY_NOT_EXISTS when the ME entry cannot be found

MPLSTP_ERR_ME_CANT_UPDATE_SERVICE_ID when an attempt is made to update the association

oam_mplstp_api_config_dm

This function enables the Delay Measurement OAM function for an MPLS-TP Maintenance Entity.

Syntax

```
s_int32_t
oam_mplstp_api_config_dm (u_int32_t vr_id, char *meg_name, char *me_name,
                                           u_int32_t interval)
```

Input Parameters

`vr_id`	VR instance ID
`meg_name`	Pointer to the MEG name.
`me_name`	The ME name
`interval`	Time delay interval for the OAM function

Output Parameters

None

Return Values

MPLSTP_API_SUCCESS when the function succeeds

© 2015 IP Infusion Inc. Proprietary

MPLSTP_ERR_VR_NOT_EXISTS when the virtual router cannot be found

MPLSTP_ERR_MEG_NAME_INVALID when `meg_name` is NULL

MPLSTP_ERR_ME_NAME_INVALID when the ME name is invalid

MPLSTP_ERR_MEG_NAME_EXCEEDS_MAX_LEN when the MEG name exceeds the maximum length

MPLSTP_ERR_ME_NAME_EXCEEDS_MAX_LEN when the ME name exceeds maximum length

MPLSTP_ERR_MEG_ENTRY_NOT_EXISTS when the MEG entry cannot be found

MPLSTP_ERR_ME_ENTRY_NOT_EXISTS when the ME entry cannot be found

MPLSTP_ERR_DM_SESSION_ALREADY_ACTIVE when the delay measurement session is already active

MPLSTP_API_FAILURE when the function fails

oam_mplstp_api_unconfig_dm

This function disables the Delay Measurement OAM function for an MPLS-TP Maintenance Entity.

Syntax

```
s_int32_t
oam_mplstp_api_unconfig_dm (u_int32_t vr_id, char *meg_name, char *me_name)
```

Input Parameters

`vr_id`	VR instance ID
`meg_name`	Pointer to the MEG name.
`me_name`	The ME name

Output Parameters

None

Return Values

MPLSTP_API_SUCCESS when the function succeeds

MPLSTP_ERR_VR_NOT_EXISTS when the virtual router cannot be found

MPLSTP_ERR_MEG_NAME_INVALID when `meg_name` is NULL

MPLSTP_ERR_ME_NAME_INVALID when the ME name is invalid

MPLSTP_ERR_MEG_NAME_EXCEEDS_MAX_LEN when the MEG name exceeds the maximum length

MPLSTP_ERR_ME_NAME_EXCEEDS_MAX_LEN when the ME name exceeds maximum length

MPLSTP_ERR_MEG_ENTRY_NOT_EXISTS when the MEG entry cannot be found

MPLSTP_ERR_ME_ENTRY_NOT_EXISTS when the ME entry cannot be found

MPLSTP_ERR_NO_ACTIVE_DM_SESSION when there are no active delay measurement session

MPLSTP_API_FAILURE when the function fails

oam_mplstp_api_config_fm

This function enables the Fault Management OAM function for an MPLS-TP Maintenance Entity.

Syntax

```
s_int32_t
oam_mplstp_api_config_fm (u_int32_t vr_id, char *meg_name,
                          char *me_name, u_int32_t interval)
```

Input Parameters

`vr_id`	VR instance ID
`meg_name`	Pointer to the MEG name.
`me_name`	The ME name
`interval`	Refresh interval for the OAM function

Output Parameters

None

Return Values

MPLSTP_API_SUCCESS when the function succeeds

MPLSTP_ERR_VR_NOT_EXISTS when the virtual router cannot be found

MPLSTP_ERR_MEG_NAME_INVALID when `meg_name` is NULL

MPLSTP_ERR_ME_NAME_INVALID – When the ME name is invalid

MPLSTP_ERR_MEG_NAME_EXCEEDS_MAX_LEN when the MEG name exceeds the maximum length

MPLSTP_ERR_ME_NAME_EXCEEDS_MAX_LEN when the ME name exceeds maximum length

MPLSTP_ERR_MEG_ENTRY_NOT_EXISTS when the MEG entry cannot be found

MPLSTP_ERR_ME_ENTRY_NOT_EXISTS when the ME entry cannot be found

MPLSTP_ERR_INVALID_FM_INTERVAL when the `interval` is invalid

MPLSTP_ERR_FM_ALREADY_ENABLED when FM is already enabled

oam_mplstp_api_unconfig_fm

This function disables the Fault Management OAM function for an MPLS-TP Maintenance Entity.

Syntax

```
s_int32_t
oam_mplstp_api_unconfig_fm (u_int32_t vr_id, char *meg_name, char *me_name)
```

Input Parameters

`vr_id`	VR instance ID
`meg_name`	Pointer to the MEG name.
`me_name`	The ME name

Output Parameters

None

`Return Values`

MPLSTP_API_SUCCESS when the function succeeds

© 2015 IP Infusion Inc. Proprietary

MPLSTP_ERR_VR_NOT_EXISTS when the virtual router cannot be found

MPLSTP_ERR_MEG_NAME_INVALID when `meg_name` is NULL

MPLSTP_ERR_ME_NAME_INVALID when the ME name is invalid

MPLSTP_ERR_MEG_NAME_EXCEEDS_MAX_LEN when the MEG name exceeds the maximum length

MPLSTP_ERR_ME_NAME_EXCEEDS_MAX_LEN when the ME name exceeds maximum length

MPLSTP_ERR_MEG_ENTRY_NOT_EXISTS when the MEG entry cannot be found

MPLSTP_ERR_ME_ENTRY_NOT_EXISTS when the ME entry cannot be found

MPLSTP_ERR_FM_NOT_ENABLED when FM is not enabled

oam_mplstp_api_config_lm

This function enables the Loss Measurement OAM function for an MPLS-TP Maintenance Entity.

Syntax

```
s_int32_t
oam_mplstp_api_config_lm (u_int32_t vr_id, char *meg_name,
                          char *me_name, u_int32_t interval)
```

Input Parameters

`vr_id`	VR instance ID
`meg_name`	Pointer to the MEG name.
`me_name`	The ME name
`interval`	Timer interval for the OAM function

Output Parameters

None

`Return Values`

MPLSTP_API_SUCCESS when the function succeeds

MPLSTP_ERR_VR_NOT_EXISTS when the virtual router cannot be found

MPLSTP_ERR_MEG_NAME_INVALID when `meg_name` is NULL

MPLSTP_ERR_ME_NAME_INVALID when the ME name is invalid

MPLSTP_ERR_MEG_NAME_EXCEEDS_MAX_LEN when the MEG name exceeds the maximum length

MPLSTP_ERR_ME_NAME_EXCEEDS_MAX_LEN when the ME name exceeds maximum length

MPLSTP_ERR_MEG_ENTRY_NOT_EXISTS when the MEG entry cannot be found

MPLSTP_ERR_ME_ENTRY_NOT_EXISTS when the ME entry cannot be found

MPLSTP_ERR_LM_SESSION_ALREADY_ACTIVE when a loss measurement session is already active.

MPLSTP_API_FAILURE when the function fails.

oam_mplstp_api_unconfig_lm

This function disables the Loss Measurement OAM function for an MPLS-TP Maintenance Entity.

Syntax

```
s_int32_t
oam_mplstp_api_unconfig_lm (u_int32_t vr_id, char *meg_name, char *me_name)
```

Input Parameters

`vr_id`	VR instance ID
`meg_name`	Pointer to the MEG name.
`me_name`	The ME name

Output Parameters

None

`Return Values`

MPLSTP_API_SUCCESS when the function succeeds

MPLSTP_ERR_VR_NOT_EXISTS when the virtual router cannot be found

MPLSTP_ERR_MEG_NAME_INVALID when `meg_name` is NULL

MPLSTP_ERR_ME_NAME_INVALID when the ME name is invalid

MPLSTP_ERR_MEG_NAME_EXCEEDS_MAX_LEN when the MEG name exceeds the maximum length

MPLSTP_ERR_ME_NAME_EXCEEDS_MAX_LEN when the ME name exceeds maximum length

MPLSTP_ERR_MEG_ENTRY_NOT_EXISTS when the MEG entry cannot be found

MPLSTP_ERR_ME_ENTRY_NOT_EXISTS when the ME entry cannot be found

MPLSTP_ERR_NO_ACTIVE_LM_SESSION when there are no active loss-measurement sessions

MPLSTP_API_FAILURE when the function fails

oam_mplstp_api_config_lock

This function enables the Loopback OAM function for an MPLS-TP Maintenance Entity.

Syntax

```
s_int32_t
oam_mplstp_api_config_loopback (u_int32_t vr_id, char *meg_name, char *me_name)
```

Input Parameters

Input Parameters

`vr_id`	VR instance ID
`meg_name`	Pointer to the MEG name.
`me_name`	The ME name

Output Parameters

None

`Return Values`

MPLSTP_API_SUCCESS when the function succeeds

MPLSTP_ERR_VR_NOT_EXISTS when the virtual router cannot be found

 © 2015 IP Infusion Inc. Proprietary

MPLSTP_ERR_MEG_NAME_INVALID when `meg_name` is NULL

MPLSTP_ERR_ME_NAME_INVALID when the ME name is invalid

MPLSTP_ERR_MEG_NAME_EXCEEDS_MAX_LEN when the MEG name exceeds the maximum length

MPLSTP_ERR_ME_NAME_EXCEEDS_MAX_LEN when the ME name exceeds maximum length

MPLSTP_ERR_MEG_ENTRY_NOT_EXISTS when the MEG entry cannot be found

MPLSTP_ERR_ME_ENTRY_NOT_EXISTS when the ME entry cannot be found

oam_mplstp_api_unconfig_lock

This function disables the Lock OAM function for an MPLS-TP Maintenance Entity.

Syntax

```
s_int32_t
oam_mplstp_api_unconfig_lock (u_int32_t vr_id, char *meg_name, char *me_name)
```

Input Parameters

Input Parameters

`vr_id`	VR instance ID
`meg_name`	Pointer to the MEG name.
`me_name`	The ME name

Output Parameters

None

`Return Values`

MPLSTP_API_SUCCESS when the function succeeds

MPLSTP_ERR_VR_NOT_EXISTS when the virtual router cannot be found

MPLSTP_ERR_MEG_NAME_INVALID when `meg_name` is NULL

MPLSTP_ERR_ME_NAME_INVALID when the ME name is invalid

MPLSTP_ERR_MEG_NAME_EXCEEDS_MAX_LEN when the MEG name exceeds the maximum length

MPLSTP_ERR_ME_NAME_EXCEEDS_MAX_LEN when the ME name exceeds maximum length

MPLSTP_ERR_MEG_ENTRY_NOT_EXISTS when the MEG entry cannot be found

MPLSTP_ERR_ME_ENTRY_NOT_EXISTS when the ME entry cannot be found

MPLSTP_ERR_LOOPBACK_ENABLED when loopback is already enabled

MPLSTP_ERR_LOCK_NOT_ENABLED when LOCK is not enabled

MPLSTP_FAILURE when the function fails

oam_mplstp_api_config_loopback

This function enables the Lock OAM function for an MPLS-TP Maintenance Entity.

Syntax

```
s_int32_t
oam_mplstp_api_config_lock (u_int32_t vr_id, char *meg_name,
                            char *me_name, u_int32_t interval)
```

Input Parameters

`vr_id`	VR instance ID
`meg_name`	Pointer to the MEG name.
`me_name`	The ME name
`interval`	Refresh interval for the OAM function

Output Parameters

None

`Return Values`

MPLSTP_API_SUCCESS when the function succeeds

MPLSTP_ERR_VR_NOT_EXISTS when the virtual router cannot be found

MPLSTP_ERR_MEG_NAME_INVALID when `meg_name` is NULL

MPLSTP_ERR_ME_NAME_INVALID when the ME name is invalid

MPLSTP_ERR_MEG_NAME_EXCEEDS_MAX_LEN when the MEG name exceeds the maximum length

MPLSTP_ERR_ME_NAME_EXCEEDS_MAX_LEN when the ME name exceeds maximum length

MPLSTP_ERR_MEG_ENTRY_NOT_EXISTS when the MEG entry cannot be found

MPLSTP_ERR_ME_ENTRY_NOT_EXISTS when the ME entry cannot be found

MPLSTP_ERR_INVALID_LI_INTERVAL when LI interval entered is invalid

MPLSTP_ERR_LOCK_ALREADY_ENABLED when lock is already enabled

MPLSTP_FAILURE when the function fails

oam_mplstp_api_unconfig_loopback

This function disables the Loopback OAM function for an MPLS-TP Maintenance Entity.

Syntax

```
s_int32_t
oam_mplstp_api_unconfig_loopback (u_int32_t vr_id, char *meg_name, char *me_name)
```

Input Parameters

`vr_id`	VR instance ID
`meg_name`	Pointer to the MEG name.
`me_name`	The ME name

Output Parameters

None

`Return Values`

© 2015 IP Infusion Inc. Proprietary

MPLSTP_API_SUCCESS when the function succeeds

MPLSTP_ERR_VR_NOT_EXISTS when the virtual router cannot be found

MPLSTP_ERR_MEG_NAME_INVALID when `meg_name` is NULL

MPLSTP_ERR_ME_NAME_INVALID when the ME name is invalid

MPLSTP_ERR_MEG_NAME_EXCEEDS_MAX_LEN when the MEG name exceeds the maximum length

MPLSTP_ERR_ME_NAME_EXCEEDS_MAX_LEN when the ME name exceeds maximum length

MPLSTP_ERR_MEG_ENTRY_NOT_EXISTS when the MEG entry cannot be found

MPLSTP_ERR_ME_ENTRY_NOT_EXISTS when the ME entry cannot be found

MPLSTP_ERR_LOOPBACK_NOT_ENABLED when loopback is not enabled

oam_mplstp_api_config_bfd

This function configures BFD session for an MPLS-TP Maintenance Entity.

Syntax

```
s_int32_t
oam_mplstp_api_config_bfd (u_int32_t vr_id, char *meg_name, char *me_name,
                           u_int32_t min_tx, u_int32_t min_rx)
```

Input Parameters

`vr_id`	VR instance ID
`meg_name`	Pointer to the MEG name.
`me_name`	The ME name
`min_tx`	Transmit Interval in milliseconds
`min_rx`	Reception Interval in milliseconds

Output Parameters

None

`Return Values`

MPLSTP_API_SUCCESS when the function succeeds

MPLSTP_ERR_VR_NOT_EXISTS when the virtual router cannot be found

MPLSTP_ERR_MEG_NAME_INVALID when `meg_name` is NULL

MPLSTP_ERR_ME_NAME_INVALID when the ME name is invalid

MPLSTP_ERR_MEG_NAME_EXCEEDS_MAX_LEN when the MEG name exceeds the maximum length

MPLSTP_ERR_ME_NAME_EXCEEDS_MAX_LEN when the ME name exceeds maximum length

MPLSTP_ERR_MEG_ENTRY_NOT_EXISTS when the MEG entry cannot be found

MPLSTP_ERR_ME_ENTRY_NOT_EXISTS when the ME entry cannot be found

MPLSTP_ERR_BFD_TX_RX_ZERO when the BFD Tx and RX intervals entered are 0

MPLSTP_ERR_BFD_INVALID_INTERVAL when the BFD interval value entered is invalid

oam_mplstp_api_unconfig_bfd

This function unconfigures BFD session for an MPLS-TP Maintenance Entity.

Syntax

```
s_int32_t
oam_mplstp_api_unconfig_bfd (u_int32_t vr_id, char *meg_name, char *me_name)
```

Input Parameters

`vr_id`	VR instance ID
`meg_name`	Pointer to the MEG name.
`me_name`	The ME name

Output Parameters

None

Return Values

MPLSTP_API_SUCCESS when the function succeeds

MPLSTP_ERR_VR_NOT_EXISTS when the virtual router cannot be found

MPLSTP_ERR_MEG_NAME_INVALID when `meg_name` is NULL

MPLSTP_ERR_ME_NAME_INVALID when the ME name is invalid

MPLSTP_ERR_MEG_NAME_EXCEEDS_MAX_LEN when the MEG name exceeds the maximum length

MPLSTP_ERR_ME_NAME_EXCEEDS_MAX_LEN when the ME name exceeds maximum length

MPLSTP_ERR_MEG_ENTRY_NOT_EXISTS when the MEG entry cannot be found

MPLSTP_ERR_ME_ENTRY_NOT_EXISTS when the ME entry cannot be found

oam_mplstp_api_config_bfd

This function configures BFD session for an MPLS-TP Maintenance Entity.

Syntax

```
s_int32_t
oam_mplstp_api_config_bfd (u_int32_t vr_id, char *meg_name, char *me_name,
                           u_int32_t min_tx, u_int32_t min_rx)
```

Input Parameters

`vr_id`	VR instance ID
`meg_name`	Pointer to the MEG name.
`me_name`	The ME name
`min_tx`	Transmit Interval in milliseconds
`min_rx`	Reception Interval in milliseconds

Output Parameters

None

© 2015 IP Infusion Inc. Proprietary

Return Values

MPLSTP_API_SUCCESS when the function succeeds

oam_mplstp_api_itut_meg_create

This function creates an MPLS-TP Maintenance Entity Group (if not already created).

Syntax

```
s_int32_t
oam_mplstp_api_itut_meg_create (u_int32_t vr_id, char *meg_name,
                                u_int8_t meg_level)
```

Input Parameters

vr_id	VR instance ID
meg_name	Pointer to the MEG name.
me_level	The MEG level

Output Parameters

None

Return Values

MPLSTP_API_SUCCESS when the function succeeds

MPLSTP_API_FAILURE when the function fails

MPLSTP_ERR_VR_NOT_EXISTS when the virtual router cannot be found

MPLSTP_ERR_MEG_NAME_INVALID when meg_name is NULL

MPLSTP_ERR_MEG_NAME_EXCEEDS_MAX_LEN when the MEG name exceeds the maximum length

MPLSTP_ERR_MEG_CREATE_FAILED when MEG creation fails

MPLSTP_ERR_MEG_NAME_EXCEEDS_MAX_LEN when the MEG name exceeds the maximum length

MPLSTP_ERR_MEG_UMC_EXCEEDS_MAX_LEN when the MEG name exceeds the maximum length

MPLSTP_ERR_MEG_LEVEL_UPDATE_NOT_POSSIBLE when the MEG already exists with a different me_level

oam_mplstp_api_itut_meg_delete

This function deletes the ITUT MPLS-TP Maintenance Entity Group if present and does not have an ME.

Syntax

```
s_int32_t
oam_mplstp_api_itut_meg_delete (u_int32_t vr_id, char *meg_name)
```

Input Parameters

vr_id	VR instance ID
meg_name	Pointer to the MEG name.

Output Parameters

None

Return Values

MPLSTP_API_SUCCESS when the function succeeds

MPLSTP_ERR_VR_NOT_EXISTS when the virtual router cannot be found

MPLSTP_ERR_MEG_NAME_INVALID when `meg_name` is NULL

MPLSTP_ERR_MEG_NAME_EXCEEDS_MAX_LEN when the MEG name exceeds the maximum length

MPLSTP_ERR_MEG_UMC_EXCEEDS_MAX_LEN when the MEG name exceeds the maximum length

MPLSTP_ERR_MEG_ENTRY_NOT_EXISTS when the MEG entry cannot be found

MPLSTP_ERR_ME_ENTRY_EXISTS when an maintenance entity exists in the MEG

MPLSTP_ERR_INVALID_MEG_TYPE when the MEG is not an ITU type

oam_mplstp_api_itut_set_meg_service_type

This function sets the service type of MPLS-TP Maintenance Entity Group. Update is not allowed if an ME exist.

Syntax

```
s_int32_t
oam_mplstp_api_itut_set_meg_service_type (u_int32_t vr_id, char *meg_name,
                                          u_int8_t service_type)
```

Input Parameters

`vr_id`	VR instance ID
`meg_name`	Pointer to the MEG name.
`service_type`	Service type, including tunnel, VC, or datalink

Output Parameters

None

Return Values

MPLSTP_API_SUCCESS when the function succeeds

MPLSTP_ERR_VR_NOT_EXISTS when the virtual router cannot be found

MPLSTP_ERR_MEG_NAME_INVALID when `meg_name` is NULL

MPLSTP_ERR_MEG_NAME_EXCEEDS_MAX_LEN when the MEG name exceeds the maximum length

MPLSTP_ERR_MEG_ENTRY_NOT_EXISTS when the MEG entry cannot be found

MPLSTP_API_ERR_CANT_MODIFY_SERVICE_TYPE

MPLSTP_ERR_INVALID_MEG_TYPE when the MEG is not an ITU type

MPLSTP_ERR_ME_ENTRY_EXISTS when an maintenance entity exists in the MEG

oam_mplstp_api_itut_mp_create

This function creates an MPLS-TP Maintenance Entity for an MPLS-TP Maintenance Entity Group.

© 2015 IP Infusion Inc. Proprietary

Syntax

```
s_int32_t
oam_mplstp_api_itut_mp_create (u_int32_t vr_id, char *meg_name,
                               u_int16_t mp_id, u_int8_t mp_type)
```

Input Parameters

`vr_id`	VR instance ID
`meg_name`	Pointer to the MEG name.
`mp_id`	Integer containing the MP ID.
`mp_type`	Specifies whether the MP type, either MEP or MIP.

Output Parameters

None

Return Values

MPLSTP_API_SUCCESS when the function succeeds

MPLSTP_API_FAILURE when the function fails

MPLSTP_ERR_VR_NOT_EXISTS when the virtual router cannot be found

MPLSTP_ERR_MEG_NAME_INVALID when `meg_name` is NULL

MPLSTP_ERR_MEG_NAME_EXCEEDS_MAX_LEN when the MEG name exceeds the maximum length

MPLSTP_ERR_MEG_ENTRY_NOT_EXISTS when the MEG entry cannot be found

MPLSTP_ERR_ME_ENTRY_ALREADY_EXISTS when the maintenance entity already exists

MPLSTP_ERR_ME_CREATE_FAILED when the ME creation process fails

MPLSTP_ERR_INVALID_MP_ID when `mp_id` is not valid

oam_mplstp_api_itut_mp_delete

This function deletes the ITUT MPLS-TP Maintenance Entity if there are no proactive OAMs configured.

Syntax

```
s_int32_t
oam_mplstp_api_itut_mp_delete (u_int32_t vr_id, char *meg_name, u_int16_t mp_id,
                               u_int8_t mp_type)
```

Input Parameters

`vr_id`	VR instance ID
`meg_name`	Pointer to the MEG name.
`mp_id`	Integer containing the MP ID.
`mp_type`	Specifies whether the MP type, either MEP or MIP.

Output Parameters

None

Return Values

MPLSTP_API_SUCCESS when the function succeeds

MPLSTP_ERR_VR_NOT_EXISTS when the virtual router cannot be found

MPLSTP_ERR_MEG_NAME_INVALID when `meg_name` is NULL

MPLSTP_ERR_MEG_NAME_EXCEEDS_MAX_LEN when the MEG name exceeds the maximum length

MPLSTP_ERR_MEG_ENTRY_NOT_EXISTS when the MEG entry cannot be found

MPLSTP_ERR_ME_ENTRY_NOT_EXISTS when the ME entry cannot be found

MPLSTP_ERR_OAM_CONFIG_EXISTS when a proactive OAM exists

MPLSTP_ERR_INVALID_MP_ID when `mp_id` is not valid

oam_mplstp_api_itut_associate_tunnel

This function associates an MPLS-TP tunnel to an MPLS-TP Maintenance Entity, if the service type configured is a tunnel. Updates are not allowed.

Syntax

```
s_int32_t
oam_mplstp_api_itut_associate_tunnel (u_int32_t vr_id,
         char *meg_name, u_int16_t mp_id, char *tnl_name)
```

Input Parameters

`vr_id`	VR instance ID
`meg_name`	Pointer to the MEG name.
`mp_id`	Integer containing the MP ID.
`tnl_name`	Pointer containing the tunnel name.

Output Parameters

None

Return Values

MPLSTP_API_SUCCESS when the function succeeds

MPLSTP_ERR_VR_NOT_EXISTS when the virtual router cannot be found

MPLSTP_ERR_MEG_NAME_INVALID when `meg_name` is NULL

MPLSTP_ERR_MEG_NAME_EXCEEDS_MAX_LEN when the MEG name exceeds the maximum length

MPLSTP_ERR_ME_TNL_NAME_INVALID when `tnl_name` is NULL

MPLSTP_ERR_ME_TNL_NAME_EXCEEDS_MAX_LEN when `tnl_name` exceeds the maximum length

MPLSTP_ERR_MEG_ENTRY_NOT_EXISTS when the MEG entry cannot be found

MPLSTP_ERR_ME_SERVICE_TYPE_NOT_MATCH when the ME service type does not match

MPLSTP_ERR_ME_ENTRY_NOT_EXISTS when the ME entry cannot be found

MPLSTP_ERR_ME_CANT_UPDATE_SERVICE_ID

© 2015 IP Infusion Inc. Proprietary

oam_mplstp_api_itut_associate_vc

This function associates an MPLS-TP VC to an MPLS-TP Maintenance Entity, if the service type configured is a VC (virtual circuit). Updates are not allowed.

Syntax

```
s_int32_t
oam_mplstp_api_itut_associate_vc (u_int32_t vr_id,
           char *meg_name, u_int16_t mp_id, u_int32_t vcid)
```

Input Parameters

`vr_id`	VR instance ID
`meg_name`	Pointer to the MEG name.
`mp_id`	Integer containing the MP ID.
`vcid`	Pointer containing the VC ID.

Output Parameters

None

Return Values

MPLSTP_API_SUCCESS when the function succeeds

MPLSTP_ERR_VR_NOT_EXISTS when the virtual router cannot be found

MPLSTP_ERR_MEG_NAME_INVALID when `meg_name` is NULL

MPLSTP_ERR_MEG_NAME_EXCEEDS_MAX_LEN when the MEG name exceeds the maximum length

MPLSTP_ERR_INVALID_MP_ID when `mp_id` is not valid

MPLSTP_ERR_ME_VC_ID_INVALID when `vcid` is zero

MPLSTP_ERR_MEG_ENTRY_NOT_EXISTS when the MEG entry cannot be found

MPLSTP_ERR_ME_SERVICE_TYPE_NOT_MATCH when the ME service type does not match

MPLSTP_ERR_ME_CREATE_FAILED when memory allocation fails

MPLSTP_ERR_ME_ENTRY_NOT_EXISTS when the ME entry cannot be found

MPLSTP_ERR_ME_CANT_UPDATE_SERVICE_ID when `mp_id` is already associated with a different `vcid`

oam_mplstp_api_itut_associate_datalink

This function associates an MPLS-TP provider-interface to an MPLS-TP Maintenance Entity. Updates are not allowed.

Syntax

```
s_int32_t
oam_mplstp_api_itut_associate_datalink (u_int32_t vr_id, char *meg_name,
                                        u_int16_t mp_id, char *if_name)
```

Input Parameters

`vr_id`	VR instance ID
`meg_name`	Pointer to the MEG name.

mp_id	Integer containing the MP ID.
*if_name	Pointer containing the interface name.

Output Parameters

None

Return Values

MPLSTP_API_SUCCESS when the function succeeds

MPLSTP_ERR_VR_NOT_EXISTS when the virtual router cannot be found

MPLSTP_ERR_MEG_NAME_INVALID when meg_name is NULL

MPLSTP_ERR_MEG_NAME_EXCEEDS_MAX_LEN when the MEG name exceeds the maximum length

MPLSTP_ERR_INVALID_MP_ID when mp_id is not valid

MPLSTP_ERR_ME_IF_NAME_INVALID when if_name is NULL

MPLSTP_ERR_ME_IF_NAME_EXCEEDS_MAX_LEN when if_name exceeds the maximum length

MPLSTP_ERR_MEG_ENTRY_NOT_EXISTS when the MEG entry cannot be found

MPLSTP_ERR_INVALID_MP_TYPE when the MEG entry is an invalid type

MPLSTP_ERR_ME_SERVICE_TYPE_NOT_MATCH when the ME service type does not match

MPLSTP_ERR_ME_CREATE_FAILED when memory allocation fails

MPLSTP_ERR_ME_ENTRY_NOT_EXISTS when the ME entry cannot be found

MPLSTP_ERR_ME_CANT_UPDATE_SERVICE_ID when mp_id is already associated with a different if_name

oam_mplstp_api_itut_rmep_create

This function creates a remote mep to an MPLS-TP Maintenance Entity. Updates are not allowed.

Syntax

```
s_int32_t
oam_mplstp_api_itut_rmep_create (u_int32_t vr_id, u_int16_t rmp_id, char *cc,
                                 char *icc, char *meg_name, u_int8_t direction,
                                 u_int16_t mp_id, char *local_meg_name)
```

Input Parameters

vr_id	VR instance ID
rmp_id	RMP ID.
*cc	Country code
*icc	Carrier code
meg_name	Pointer to the MEG name.
direction	Direction
mp_id	Integer containing the MP ID.
*local_meg_name	Pointer containing the local MEG name.

© 2015 IP Infusion Inc. Proprietary

Output Parameters

None

Return Values

CFM_API_SUCCESS when the function succeeds

MPLSTP_API_FAILURE when the function fails

MPLSTP_ERR_VR_NOT_EXISTS when the virtual router cannot be found

MPLSTP_ERR_MEG_NAME_INVALID when `meg_name` is NULL

MPLSTP_ERR_MEG_NAME_EXCEEDS_MAX_LEN when the MEG name exceeds the maximum length

MPLSTP_ERR_INVALID_MP_ID when `mp_id` or `rmp_id` is not valid

MPLSTP_ERR_CC_NAME_INVALID when `cc` is NULL, exceeds the maximum length, or cannot be converted to uppercase

MPLSTP_ERR_ICC_NAME_INVALID when `icc` is NULL, exceeds the maximum length, or cannot be converted to uppercase

MPLSTP_ERR_MEG_ENTRY_NOT_EXISTS when the MEG entry cannot be found

MPLSTP_ERR_ME_ENTRY_NOT_EXISTS when the ME entry does not exist

MPLSTP_ERR_RMEP_ENTRY_EXISTS when a remote MEP already exists

MPLSTP_ERR_RMEP_CREATE_FAILED when memory deallocation fails

MPLSTP_ERR_ME_SERVICE_TYPE_NOT_MATCH when the ME service type does not match

MPLSTP_ERR_ME_ENTRY_NOT_EXISTS when the ME entry cannot be found

oam_mplstp_api_itut_rmep_delete

This function deletes a remote mep from an MPLS-TP Maintenance Entity. Updates are not allowed.

Syntax

```
s_int32_t
oam_mplstp_api_itut_rmep_delete (u_int32_t vr_id, u_int16_t rmp_id,
                 char *cc, char *icc, char *meg_name, u_int16_t mp_id,
                  char *local_meg_name)
```

Input Parameters

`vr_id`	VR instance ID
`rmp_id`	RMP ID.
`*cc`	Country code
`*icc`	Carrier code
`meg_name`	Pointer to the MEG name.
`direction`	Direction
`mp_id`	Integer containing the MP ID.
`*local_meg_name`	Pointer containing the local MEG name.

Output Parameters

None

Return Values

CFM_API_SUCCESS when the function succeeds

CFM_API_FAILURE when the function fails

MPLSTP_ERR_VR_NOT_EXISTS when the virtual router cannot be found

MPLSTP_ERR_MEG_NAME_INVALID when `meg_name` is NULL

MPLSTP_ERR_MEG_NAME_EXCEEDS_MAX_LEN when the MEG name exceeds the maximum length

MPLSTP_ERR_INVALID_MP_ID when `mp_id` or `rmp_id` is not valid

MPLSTP_ERR_CC_NAME_INVALID when `cc` is NULL, exceeds the maximum length, or cannot be converted to uppercase

MPLSTP_ERR_ICC_NAME_INVALID when `icc` is NULL, exceeds the maximum length, or cannot be converted to uppercase

MPLSTP_ERR_MEG_ENTRY_NOT_EXISTS when the MEG entry cannot be found

MPLSTP_ERR_ME_ENTRY_NOT_EXISTS when the ME entry does not exist

MPLSTP_ERR_CC_CONFD_ON_MEP when the Continuity Check Initiator is enabled

MPLSTP_ERR_RMEP_ENTRY_NOT_EXISTS when the remote MEP does not exist

oam_mplstp_api_itut_recv_path_info

This function get the path information from NSM. The Path info can be PATH UP and will include MPLS-TP Tunnel or VC info. In addition, info can include PATH DOWN or PATH ERR - Tunnel or VC already associated with another MPLS-TP Maintenance Entity.

Syntax

```
void
oam_mplstp_api_itut_recv_path_info (struct bfd_master *bm,
                                    struct mplstp_path_info *path_info)
```

Input Parameters

`*bm`	Pointer to the BFD master.
`*path_info`	MPLS-TP path information

Output Parameters

None

Return Values

None

oam_mplstp_api_itut_config_ais

This function enables the alarm indication signal (AIS) feature.

This function implements the `alarm-indication level <0-7> interval (1|60)` command.

© 2015 IP Infusion Inc. Proprietary

Syntax

```
s_int32_t
oam_mplstp_api_itut_config_ais (u_int32_t vr_id, char *meg_name,
                                u_int16_t mp_id, u_int8_t level,
                                u_int32_t interval)
```

Input Parameters

`vr_id`	VR instance ID
`meg_name`	MEG name
`mp_id`	MP ID
`level`	Level
`interval`	Interval

Output Parameters

None

Return Values

MPLSTP_ERR_VR_NOT_EXISTS when the virtual router cannot be found

MPLSTP_ERR_MEG_ENTRY_NOT_EXISTS when the MEG entry cannot be found

MPLSTP_ERR_ME_ENTRY_NOT_EXISTS when the ME entry cannot be found

MPLSTP_ERR_INVALID_MEG_LEVEL when `level` is not valid

MPLSTP_ERR_INVALID_AIS_INTERVAL when `interval` is not valid

MPLSTP_ERR_UPDATING_AIS_INTERVAL_NOT_ALLOWED when AIS is already enabled with a different `interval`

MPLSTP_ERR_INVALID_MP_ID when `mp_id` is not valid

MPLSTP_ERR_MEG_NAME_INVALID when `meg_name` is NULL

MPLSTP_ERR_MEG_NAME_EXCEEDS_MAX_LEN when the MEG name exceeds the maximum length

MPLSTP_ERR_MEG_UMC_EXCEEDS_MAX_LEN when the MEG name exceeds the maximum length

MPLSTP_API_FAILURE when the function fails

MPLSTP_API_SUCCESS when the function succeeds

oam_mplstp_api_itut_config_mp_cc

This function enables continuity-check messaging (CCM) for a MEG.

This function implements the `continuity-check interval` command.

Syntax

```
s_int32_t
oam_mplstp_api_itut_config_mp_cc (u_int32_t vr_id, char *meg_name,
                                  u_int16_t mp_id, u_int8_t interval)
```

Input Parameters

`vr_id`	VR instance ID

`meg_name`	MEG name
`mp_id`	MP ID
`interval`	Interval

Output Parameters

None

Return Values

MPLSTP_ERR_VR_NOT_EXISTS when the virtual router cannot be found

MPLSTP_ERR_MEG_ENTRY_NOT_EXISTS when the MEG entry cannot be found

MPLSTP_ERR_ME_ENTRY_NOT_EXISTS when the ME entry cannot be found

MPLSTP_ERR_INVALID_CC_INTERVAL when `interval` is not valid

MPLSTP_ERR_INVALID_MP_ID when `mp_id` is not valid

MPLSTP_ERR_MEG_NAME_INVALID when `meg_name` is NULL

MPLSTP_ERR_MEG_NAME_EXCEEDS_MAX_LEN when the MEG name exceeds the maximum length

MPLSTP_ERR_MEG_UMC_EXCEEDS_MAX_LEN when the MEG name exceeds the maximum length

MPLSTP_API_FAILURE when the function fails

MPLSTP_API_SUCCESS when the function succeeds

oam_mplstp_api_itut_config_meg_cc

This function enables continuity-check messaging (CCM) for a MEP.

This function implements the `continuity-check interval` command.

Syntax

```
s_int32_t
oam_mplstp_api_itut_config_meg_cc (u_int32_t vr_id, char *meg_name,
                                   u_int8_t interval)
```

Input Parameters

`vr_id`	VR instance ID
`meg_name`	MEG name
`interval`	Interval

Output Parameters

None

Return Values

MPLSTP_ERR_VR_NOT_EXISTS when the virtual router cannot be found

MPLSTP_ERR_MEG_ENTRY_NOT_EXISTS when the MEG entry cannot be found

MPLSTP_ERR_INVALID_CC_INTERVAL when `interval` is not valid

MPLSTP_ERR_ME_ENTRY_NOT_EXISTS when the ME entry cannot be found

© 2015 IP Infusion Inc. Proprietary

MPLSTP_ERR_MEG_NAME_INVALID when `meg_name` is NULL

MPLSTP_ERR_MEG_NAME_EXCEEDS_MAX_LEN when the MEG name exceeds the maximum length

MPLSTP_ERR_MEG_UMC_EXCEEDS_MAX_LEN when the MEG name exceeds the maximum length

MPLSTP_API_FAILURE when the function fails

MPLSTP_API_SUCCESS when the function succeeds

oam_mplstp_api_itut_config_fng

This function enables the fault alarm feature.

This function implements the fault-alarm command.

Syntax

```
s_int32_t
oam_mplstp_api_itut_config_fng (u_int32_t vr_id, char *meg_name,
                                u_int16_t mp_id, u_int8_t priority,
                                u_int32_t alarm_time, u_int32_t reset_time)
```

Input Parameters

`vr_id`	VR instance ID
`meg_name`	MEG name
`mp_id`	MP ID
`priority`	Priority
`alarm_time`	Alarm time
`reset_time`	Reset time

Output Parameters

None

Return Values

MPLSTP_ERR_VR_NOT_EXISTS when the virtual router cannot be found

MPLSTP_ERR_MEG_ENTRY_NOT_EXISTS when the MEG entry cannot be found

MPLSTP_ERR_ME_ENTRY_NOT_EXISTS when the ME entry cannot be found

MPLSTP_ERR_CC_ENABLED when continuity checking is already enabled

MPLSTP_ERR_INAVLID_FNG_PRIORITY when `priority` is not valid

MPLSTP_ERR_INVALID_MP_ID when `mp_id` is not valid

MPLSTP_API_FAILURE when the function fails

MPLSTP_API_SUCCESS when the function succeeds

© 2015 IP Infusion Inc. Proprietary

© 2015 IP Infusion Inc. Proprietary

CHAPTER 21 MPLS-TP Linear Protection Switching API

The functions in this chapter are called by the LPS commands used to configure and manage protection switching coordination.

Functions

Function	Description
oam_api_lps_backup_meg_me_set	Sets the backup MEG and IETF ME for a protection group
oam_api_lps_backup_meg_me_unset	Removes the backup MEG and IETF ME for a protection group
oam_api_lps_backup_meg_mep_set	Sets the backup MEG and ITU-T MEP for a protection group
oam_api_lps_backup_meg_mep_unset	Removes the backup MEG and ITU-T MEP for a protection group
oam_api_lps_continual_tx_interval_set	Sets a continual transmission interval for PSC packets
oam_api_lps_debug_event	Enables debugging of PSC events
oam_api_lps_get_protection_group_from_name	Retrieves a protection group using group name
oam_api_lps_group_create	Creates a protection group
oam_api_lps_group_delete	Deletes a protection group
oam_api_lps_group_mode_set	Sets the protection group mode
oam_api_lps_group_scheme_set	Sets the protection scheme tor the protection group
oam_api_lps_hold_off_timer_set	Sets a hold-off timer for protection events
oam_api_lps_lockout	Initiates lockout of protection
oam_api_lps_no_debug_event	Disables debugging of PSC events
oam_api_lps_lockout	Disables lockout of protection
oam_api_lps_no_switchover	Disables on-demand switchover
oam_api_lps_primary_meg_me_set	Sets the primary MEG and IETF ME for a protection group
oam_api_lps_primary_meg_me_unset	Removes the primary MEG and IETF ME for a protection group
oam_api_lps_primary_meg_mep_set	Sets the primary MEG and ITU-T MEP for a protection group
oam_api_lps_primary_meg_mep_unset	Removes the primary MEG and ITU-T MEP for a protection group
oam_api_lps_rapid_tx_interval_set	Sets a rapid transmission interval for PSC packets

Function	Description
oam_api_lps_switchover	Initiates on-demand switchover
oam_api_lps_wtr_timer_set	Sets a wait-to-restore timer for a protection group

oam_api_lps_backup_meg_me_set

This function sets the backup MEG (Maintenance Entity Group) and IETF ME (Maintenance Entity) for a protection group.

Syntax

```
s_int32_t
oam_api_lps_backup_meg_me_set(u_int32_t vr_id, char *grp_name_str,
                              char *meg_name, char *me_name)
```

Input Parameters

vr_id	VR instance ID
grp_name_str	The protection-group name
meg_name	MEG name; maximum 48 characters
me_name	IETF ME identifier; maximum 48 characters

Output Parameters

None

Return Values

OAM_API_ERR_ME_ALREADY_ASSOCIATED when the protection-group already has ME associated

OAM_API_ERR_PG_ALREADY_HAS_BACKUP_ME when the protection-group already has a backup ME associated.

OAM_API_ERR_MEG_ENTRY_NOT_EXISTS when the MEG entry does not exist

OAM_API_ERR_ME_ENTRY_NOT_EXISTS when the ME entry does not exist

OAM_API_SET_ERROR_PSC_GRP_LOOKUP when the protection-group lookup fails

OAM_API_ME_ALREADY_PROTECTED when the ME entry is already under protection-group

OAM_API_ERR_MEG_SERVICE_TYPE_NOT_TUNNEL when the service-type of MEG is not tunnel

OAM_API_LPS_SET_ERROR when the function fails due to internal errors such as a memory failure

OAM_API_LPS_SET_SUCCESS when the function succeeds

oam_api_lps_backup_meg_me_unset

This function removes the backup MEG (Maintenance Entity Group) and IETF ME (Maintenance Entity) for a protection group.

Syntax

```
s_int32_t
oam_api_lps_backup_meg_me_unset(u_int32_t vr_id, char *grp_name_str,
```


© 2015 IP Infusion Inc. Proprietary

```
                                char *meg_name, char *me_name)
```

Input Parameters

`vr_id`	VR instance ID
`grp_name_str`	The protection-group name
`meg_name`	MEG name; maximum 48 characters
`me_name`	IETF ME identifier; maximum 48 characters

Output Parameters

None

Return Values

OAM_API_SET_ERROR_PSC_GRP_LOOKUP when the protection-group lookup fails

OAM_API_ERR_MEG_ENTRY_NOT_EXISTS when the MEG entry does not exist

OAM_API_ERR_ME_ENTRY_NOT_EXISTS when the ME entry does not exist

OAM_API_ERR_ME_ENTRY_NOT_IN_PG when the ME entry is not associated with the given protection group

OAM_API_ERR_ME_ENTRY_PG_MISMATCH_ERROR when the ME entry is not associated with a protection-group

OAM_API_LPS_SET_ERROR when the function fails due to internal errors such as a memory failure

OAM_API_LPS_SET_SUCCESS when the function succeeds

oam_api_lps_backup_meg_mep_set

This function sets the backup MEG (Maintenance Entity Group) and ITU-T MEP (MEG End Point) for a protection group.

Syntax

```
s_int32_t
oam_api_lps_backup_meg_mep_set (u_int32_t vr_id, char *grp_name_str,
                                char *meg_name, u_int16_t mp_id)
```

Input Parameters

`vr_id`	VR instance ID
`grp_name_str`	The protection group name
`meg_name`	MEG name; maximum 48 characters
`mp_id`	ITU-T MEP identifier

Output Parameters

None

Return Values

OAM_API_SET_ERROR_PSC_GRP_LOOKUP when the protection group lookup fails

OAM_API_SET_ERR_INVALID_MEG_NAME when `meg_name` is not valid

OAM_API_SET_ERR_MEG_NAME_EXCEEDS_MAX_LEN when `meg_name` is too long

OAM_API_ERR_MEG_ENTRY_NOT_EXISTS when `meg_name` does not exist

OAM_API_ERR_ITUT_OAM_ON_IETF_MEG when the MEG type is not ITU-T

OAM_API_SET_ERROR_INVALID_MP_ID when `mp_id` is not valid

OAM_API_ERR_MP_ENTRY_NOT_EXISTS when the `mp_id` lookup fails

OAM_API_ERR_MEG_SERVICE_TYPE_NOT_TUNNEL when the service-type of MEG is not tunnel

OAM_API_SET_ERROR_MEG_TYPE_MISMATCH when the protection group type is IETF

OAM_API_ERR_BIDIR_ONE_PLUS_ONE_NOT_SUPPORTED when there is an internal error

OAM_API_ERR_BIDIR_NON_REVERTIVE_NOT_SUPPORTED when there is an internal error

OAM_API_ERR_COROUTED_UNIDIR_NOT_SUPPORTED when there is an internal error

OAM_API_ERR_PG_ALREADY_HAS_BACKUP_MP when the protection group already has an associated backup MEP

OAM_API_ERR_MP_ALREADY_ASSOCIATED when the MEP is already configured as a primary MEG

OAM_API_MP_ALREADY_PROTECTED when the MEP is already part of a protection group

OAM_API_LPS_SET_ERROR when there is an internal error

OAM_API_LPS_SET_SUCCESS when the function succeeds

oam_api_lps_backup_meg_mep_unset

This function removes the backup MEG (Maintenance Entity Group) and ITU-T MEP (MEG End Point) for a protection group.

Syntax

```
s_int32_t
oam_api_lps_backup_meg_mep_unset (u_int32_t vr_id, char *grp_name_str,
                                  char *meg_name, u_int16_t mp_id)
```

Input Parameters

`vr_id`	VR instance ID
`grp_name_str`	The protection group name
`meg_name`	MEG name; maximum 48 characters
`mp_id`	ITU-T MEP identifier

Output Parameters

None

Return Values

OAM_API_SET_ERROR_PSC_GRP_LOOKUP when the protection group lookup fails

OAM_API_ARR_ITUT_PG_DOES_NOT_EXISTS when the protection group type is not ITU-T

OAM_API_LPS_SET_ERROR when `meg_name` is not valid

OAM_API_ERR_MEG_ENTRY_NOT_EXISTS when the MEG entry does not exist

OAM_API_SET_ERROR_INVALID_MP_ID when `mp_id` is not valid

OAM_API_ERR_MP_ENTRY_NOT_EXISTS when the `mp_id` lookup fails

 © 2015 IP Infusion Inc. Proprietary

OAM_API_ERR_MP_ENTRY_NOT_IN_PG when the MP entry is not associated with the protection group

OAM_API_ERR_MP_ENTRY_PG_MISMATCH_ERROR when the MP entry is not associated with the protection group

OAM_API_LPS_SET_SUCCESS when the function succeeds

oam_api_lps_continual_tx_interval_set

This function sets the continual-transmit interval for the protection-group identified by protection-group name

Syntax

```
s_int32_t
oam_api_lps_continual_tx_interval_set (u_int32_t vr_id, char *grp_name_str,
                                       u_int32_t continual_tx)
```

Input Parameters

`vr_id`	VR instance ID
`grp_name_str`	The protection-group name
`continual_tx`	Continual-transmit interval in seconds; input range is <1-20>

Output Parameters

None

Return Values

OAM_API_LPS_SET_ERROR when the function fails due to internal errors such as a memory failure

OAM_API_SET_ERROR_PSC_GRP_LOOKUP when the protection-group lookup fails

OAM_API_LPS_SET_SUCCESS when the function succeeds

oam_api_lps_debug_event

This function enables event debugging .

Syntax

```
s_int32_t
oam_api_lps_debug_event (u_int32_t vr_id, lps_debug_event debug_event)
```

Input Parameters

`vr_id`	VR instance ID
`event`	Variable of type `lps_debug_event` indicating the event type

Output Parameters

None

Return Values

OAM_API_SET_ERR_VR_NOT_EXIST when the `vr_id` is not found

OAM_API_LPS_SET_ERROR when the function fails due to internal errors such as memory failure

OAM_API_LPS_SET_SUCCESS when the function succeeds

oam_api_lps_get_protection_group_from_name

This function gets protection-group details based on the protection-group name or returns a pointer to a protection-group configured with the same name.

Syntax

```
s_int32_t
oam_api_lps_get_protection_group_from_name (u_int32_t vr_id, char *grp_name_str,
                                            struct mpls_tp_lps_group **protection_grp)
```

Input Parameters

`vr_id`	VR instance ID
`grp_name_str`	The protection-group name

Output Parameters

`protection_grp`	Pointer to protection group if protection-group name exists

Return Values

OAM_API_SET_ERR_INVALID_VALUE when the value is invalid

OAM_API_SET_ERR_PSC_GRP_NAME_EXCEEDS_MAX_LEN when the protection group-name exceeds maximum length

OAM_API_SET_ERR_VR_NOT_EXIST when `vr_id` is not found

OAM_API_SET_ERROR_PSC_GRP_LOOKUP when the protection-group lookup fails

OAM_API_LPS_SET_ERROR when the function fails due to internal errors such as a memory failure

OAM_API_LPS_GET_SUCCESS when the function succeeds

oam_api_lps_group_create

This function creates a protection-group based on the protection-group name or returns a protection-group configured with the same name.

Syntax

```
s_int32_t
oam_api_lps_group_create (u_int32_t vr_id, char *grp_name_str,
                          struct mpls_tp_lps_group **protection_grp)
```

Input Parameters

`vr_id`	VR instance ID
`grp_name_str`	The protection-group name

Output Parameters

`protection_grp`	Pointer to protection group

© 2015 IP Infusion Inc. Proprietary

Return Values

OAM_API_SET_ERR_INVALID_VALUE when the value is invalid\

OAM_API_SET_ERR_PSC_GRP_NAME_EXCEEDS_MAX_LEN when the protection group-name exceeds maximum length

OAM_API_SET_ERR_VR_NOT_EXIST when `vr_id` is not found

OAM_API_LPS_SET_ERROR when the function fails

OAM_API_SET_ERROR_PSC_GRP_CREATE when the function fails because of an internal error such as a memory failure

OAM_API_SET_ERROR_PSC_GRP_INSERT when the function fails because of an internal error such as a insertion failure

OAM_API_LPS_SET_ERROR when the function fails due to internal errors such as a memory failure

OAM_API_LPS_SET_SUCCESS when the function succeeds

oam_api_lps_group_delete

This function deletes a protection-group based on the protection-group name.

Syntax

```
s_int32_t
oam_api_lps_group_delete (u_int32_t vr_id, char *grp_name_str)
```

Input Parameters

`vr_id`	VR instance ID
`grp_name_str`	The protection-group name

Output Parameters

None

Return Values

OAM_API_SET_ERR_INVALID_VALUE when the value is invalid

OAM_API_SET_ERR_PSC_GRP_NAME_EXCEEDS_MAX_LEN when the protection group-name exceeds maximum length

OAM_API_SET_ERR_VR_NOT_EXIST when the `vr_id` is not found

OAM_API_SET_ERROR_PSC_GRP_LOOKUP when the protection-group lookup fails

OAM_API_LPS_SET_ERROR when the function fails due to internal errors such as a memory failure

OAM_API_LPS_SET_SUCCESS when the function succeeds

oam_api_lps_group_mode_set

This function sets the reversion mode of the protection-group identified by protection-group name

Syntax

```
s_int32_t
oam_api_lps_group_mode_set (u_int32_t vr_id, char *grp_name_str, rever_mode type)
```

Input Parameters

`vr_id`	VR instance ID
`grp_name_str`	The protection-group name
`type`	Protection mode type, one of
	`REVERTIVE`
	`NON-REVERTIVE`

Output Parameters

None

Return Values

OAM_API_SET_ERR_INVALID_REVER_MODE when the reversion-mode is invalid

OAM_API_SET_ERROR_PSC_GRP_LOOKUP when the protection-group lookup fails

OAM_API_LPS_SET_ERROR when the function fails due to internal errors such as a memory failure

OAM_API_LPS_SET_SUCCESS when the function succeeds

oam_api_lps_group_scheme_set

This function sets the protection scheme of the protection-group identified by protection-group name

Syntax

```
s_int32_t
oam_api_lps_group_scheme_set (u_int32_t vr_id, char *grp_name_str, lps_scheme scheme)
```

Input Parameters

`vr_id`	VR instance ID
`grp_name_str`	The protection-group name
`scheme`	Protection mode scheme:
	`UNIDIRECTIONAL PERMANENT`
	`UNIDIRECTIONAL SELECTOR`
	`BIDIRECTIONAL PERMANENT`
	`BIDIRECTIONAL SELECTOR`

Output Parameters

None

Return Values

OAM_API_SET_ERR_INVALID_LPS_SCHEME when the protection mode scheme is invalid

OAM_API_SET_ERR_VR_NOT_EXIST when the `vr_id` is not found

© 2015 IP Infusion Inc. Proprietary

OAM_API_SET_ERROR_PSC_GRP_LOOKUP when the protection-group lookup fails

OAM_API_LPS_SET_ERROR when the function fails due to internal errors such as a memory failure

OAM_API_LPS_SET_SUCCESS when the function succeeds

oam_api_lps_hold_off_timer_set

This function sets the HOLD-OFF timer value of the protection-group identified by protection-group name.

Syntax

```
s_int32_t
oam_api_lps_hold_off_timer_set (u_int32_t vr_id, char *grp_name_str,
                                u_int32_t hold_off_timer)
```

Input Parameters

`vr_id`	VR instance ID
`grp_name_str`	The protection-group name
`hold_off_time`	Timer value in seconds; input range is <0-720>

Output Parameters

None

Return Values

OAM_API_SET_ERR_INVALID_HOLD_OFF_TIMER_VALUE when the hold-off-timer value is invalid

OAM_API_SET_ERROR_PSC_GRP_LOOKUP when the protection-group lookup fails

OAM_API_LPS_SET_SUCCESS when the function succeeds

oam_api_lps_lockout

This function triggers a lockout event identified by protection-group name.

Syntax

```
s_int32_t
oam_api_lps_lockout (u_int32_t vr_id, char *grp_name_str)
```

Input Parameters

`vr_id`	VR instance ID
`grp_name_str`	The protection-group name

Output Parameters

None

Return Values

OAM_API_SET_ERROR_PSC_GRP_LOOKUP when the protection-group lookup fails

OAM_API_SET_ERROR_PRIMARY_ME_NOT_SET when the primary ME is not configured for protection-group

OAM_API_SET_ERROR_BACKUP_ME_NOT_SET when the backup ME is not configured for protection-group

OAM_API_LOCKOUT_ALREADY_EXISTS when the current state is already lockout

OAM_API_LPS_SET_ERROR when the function fails due to internal errors such as a memory failure

OAM_API_LPS_SET_SUCCESS when the function succeeds

oam_api_lps_no_debug_event

This function disables event debugging.

Syntax

```
s_int32_t
oam_api_lps_no_debug_event (u_int32_t vr_id,
                            lps_debug_event debug_event)
```

Input Parameters

`vr_id`	VR instance ID
`event`	Variable of type `lps_debug_event` indicating the event type

Output Parameters

None

Return Values

OAM_API_SET_ERR_VR_NOT_EXIST when the `vr_id` is not found

OAM_API_LPS_SET_ERROR when the function fails due to internal errors such as memory failure

OAM_API_LPS_SET_SUCCESS when the function succeeds

oam_api_lps_no_lockout

This function clears a lockout event.

Syntax

```
s_int32_t
oam_api_lps_no_lockout (u_int32_t vr_id, char *grp_name_str)
```

Input Parameters

`vr_id`	VR instance ID
`grp_name_str`	The protection-group name

Output Parameters

None

Return Values

OAM_API_SET_ERROR_PSC_GRP_LOOKUP when the protection-group lookup fails

OAM_API_SET_ERROR_PRIMARY_ME_NOT_SET when the primary ME is not configured for protection-group

OAM_API_SET_ERROR_BACKUP_ME_NOT_SET when the backup ME is not configured for protection-group

© 2015 IP Infusion Inc. Proprietary

OAM_API_FSM_NOT_IN_LOP when the current state is not lockout

OAM_API_SET_ERROR_MODE_SCHEME_MISMATCH when there is a mode or scheme mismatch

OAM_API_LPS_SET_ERROR when the function fails due to internal errors such as a memory failure

OAM_API_LPS_SET_SUCCESS when the function succeeds

oam_api_lps_no_switchover

This function clears a switchover event identified by protection-group name.

Syntax

```
s_int32_t
oam_api_lps_no_switchover (u_int32_t vr_id, char *grp_name_str,
                           lps_switch_over switch_over)
```

Input Parameters

`vr_id`	VR instance ID
`grp_name_str`	The protection-group name
`switch_over`	Identifies a switchover event

Output Parameters

None

Return Values

OAM_API_SET_ERROR_PSC_GRP_LOOKUP when the protection-group lookup fails

OAM_API_SET_ERROR_PRIMARY_ME_NOT_SET when the primary ME is not configured for protection-group

OAM_API_SET_ERROR_BACKUP_ME_NOT_SET when the backup ME is not configured for protection-group

OAM_API_FSM_NOT_IN_FORCED_SWITCH_STATE when the current state is not forced switchover

OAM_API_FSM_NOT_IN_MANUAL_SWITCH_STATE when the current state is not manual switchover

OAM_API_SET_ERROR_MODE_SCHEME_MISMATCH when there is a mode or scheme mismatch

OAM_API_LPS_SET_ERROR when the function fails due to internal errors such as a memory failure

OAM_API_LPS_SET_SUCCESS when the function succeeds

oam_api_lps_primary_meg_me_set

This function configures a primary MEG (Maintenance Entity Group) and IETF ME (Maintenance Entity) for a protection group.

Syntax

```
s_int32_t
oam_api_lps_primary_meg_me_set (u_int32_t vr_id, char *grp_name_str,
                                char *meg_name, char *me_name)
```

Input Parameters

`vr_id`	VR instance ID

`grp_name_str`	The protection-group name
`meg_name`	MEG name; maximum 48 characters
`me_name`	IETF ME identifier; maximum 48 characters

Output Parameters

None

Return Values

OAM_API_SET_ERROR_PSC_GRP_LOOKUP when the protection-group lookup fails

OAM_API_ERR_MEG_ENTRY_NOT_EXISTS when the MEG entry does not exist

OAM_API_ERR_ME_ENTRY_NOT_EXISTS when the ME entry does not exist

OAM_API_ERR_ME_ENTRY_NOT_IN_PG when the ME entry is not associated with the given protection group

OAM_API_ERR_ME_ENTRY_PG_MISMATCH_ERROR when the given ME entry is not associated with the protection-group

OAM_API_LPS_SET_ERROR when the function fails due to internal errors such as a memory failure

OAM_API_LPS_SET_SUCCESS when the function succeeds

oam_api_lps_primary_meg_me_unset

This function removes the primary MEG (Maintenance Entity Group) and IETF ME (Maintenance Entity) for a protection group.

Syntax

```
s_int32_t
oam_api_lps_primary_meg_me_unset (u_int32_t vr_id, char *grp_name_str,
                                  char *meg_name, char *me_name)
```

Input Parameters

`vr_id`	VR instance ID
`grp_name_str`	The protection-group name
`meg_name`	MEG name; maximum 48 characters
`me_name`	IETF ME identifier; maximum 48 characters

Output Parameters

None

Return Values

OAM_API_SET_ERROR_PSC_GRP_LOOKUP when the protection-group lookup fails

OAM_API_ERR_MEG_ENTRY_NOT_EXISTS when the MEG entry does not exist

OAM_API_ERR_ME_ENTRY_NOT_EXISTS when the ME entry does not exist

OAM_API_ERR_ME_ENTRY_NOT_IN_PG when the ME entry is not associated with the given protection group

OAM_API_ERR_ME_ENTRY_PG_MISMATCH_ERROR when the given ME entry is not associated with the protection-group

 © 2015 IP Infusion Inc. Proprietary

OAM_API_LPS_SET_ERROR when the function fails due to internal errors such as a memory failure

OAM_API_LPS_SET_SUCCESS when the function succeeds

oam_api_lps_primary_meg_mep_set

This function sets a primary MEG (Maintenance Entity Group) and ITU-T MEP (MEG End Point) for a protection group.

Syntax

```
s_int32_t
oam_api_lps_primary_meg_mep_set(u_int32_t vr_id, char *grp_name_str,
                                char *meg_name, u_int16_t mp_id)
```

Input Parameters

`vr_id`	VR instance ID
`grp_name_str`	The protection group name
`meg_name`	MEG name; maximum 48 characters
`mp_id`	ITU-T MEP identifier

Output Parameters

None

Return Values

OAM_API_SET_ERROR_PSC_GRP_LOOKUP when the protection group lookup fails

OAM_API_SET_ERR_INVALID_MEG_NAME when `meg_name` is not valid

OAM_API_SET_ERR_MEG_NAME_EXCEEDS_MAX_LEN when `meg_name` is too long

OAM_API_ERR_MEG_ENTRY_NOT_EXISTS when `meg_name` does not exist

OAM_API_ERR_ITUT_OAM_ON_IETF_MEG when the MEG type is not ITU-T

OAM_API_SET_ERROR_INVALID_MP_ID when `mp_id` is not valid

OAM_API_ERR_MP_ENTRY_NOT_EXISTS when the `mp_id` lookup fails

OAM_API_ERR_MEG_SERVICE_TYPE_NOT_TUNNEL when the service-type of MEG is not tunnel

OAM_API_SET_ERROR_MEG_TYPE_MISMATCH when the protection group type is IETF

OAM_API_ERR_BIDIR_ONE_PLUS_ONE_NOT_SUPPORTED when there is an internal error

OAM_API_ERR_BIDIR_NON_REVERTIVE_NOT_SUPPORTED when there is an internal error

OAM_API_ERR_COROUTED_UNIDIR_NOT_SUPPORTED when there is an internal error

OAM_API_ERR_PG_ALREADY_HAS_PRIMARY_MP when the protection group already has an associated primary MEP

OAM_API_ERR_MP_ALREADY_ASSOCIATED when the MEP is already configured as a primary MEG

OAM_API_MP_ALREADY_PROTECTED when the MEP is already part of a protection group

OAM_API_LPS_SET_ERROR when there is an internal error

OAM_API_LPS_SET_SUCCESS when the function succeeds

oam_api_lps_primary_meg_mep_unset

This function removes a primary MEG (Maintenance Entity Group) and ITU-T MEP (MEG End Point) for a protection group.

Syntax

```
s_int32_t
oam_api_lps_primary_meg_mep_unset (u_int32_t vr_id, char *grp_name_str,
                                   char *meg_name, u_int16_t mp_id)
```

Input Parameters

`vr_id`	VR instance ID
`grp_name_str`	The protection group name
`meg_name`	MEG name; maximum 48 characters
`mp_id`	ITU-T MEP identifier

Output Parameters

None

Return Values

OAM_API_SET_ERROR_PSC_GRP_LOOKUP when the protection group lookup fails

OAM_API_ARR_ITUT_PG_DOES_NOT_EXISTS when the protection group type is not ITU-T

OAM_API_LPS_SET_ERROR when `meg_name` is not valid

OAM_API_ERR_MEG_ENTRY_NOT_EXISTS when the MEG entry does not exist

OAM_API_SET_ERROR_INVALID_MP_ID when `mp_id` is not valid

OAM_API_ERR_MP_ENTRY_NOT_EXISTS when the `mp_id` lookup fails

OAM_API_ERR_MP_ENTRY_NOT_IN_PG when the MP entry is not associated with the protection group

OAM_API_ERR_MP_ENTRY_PG_MISMATCH_ERROR when the MP entry is not associated with the protection group

OAM_API_LPS_SET_SUCCESS when the function succeeds

oam_api_lps_rapid_tx_interval_set

This function sets the rapid-transmit interval for the protection-group identified by protection-group name

Syntax

```
s_int32_t
oam_api_lps_rapid_tx_interval_set (u_int32_t vr_id, char *grp_name_str,
                                   float32_t rapid_tx)
```

Input Parameters

`vr_id`	VR instance ID
`grp_name_str`	The protection-group name
`rapid_tx`	Rapid-transmit interval in microseconds; input range is <1000-1000000>

 © 2015 IP Infusion Inc. Proprietary

Output Parameters

None

Return Values

OAM_API_SET_ERR_INVALID_RAPID_TX_VALUE when the rapid timer value is invalid

OAM_API_SET_ERROR_PSC_GRP_LOOKUP when the protection-group lookup fails

OAM_API_LPS_SET_ERROR when the function fails due to internal errors such as a memory failure

OAM_API_LPS_SET_SUCCESS when the function succeeds

oam_api_lps_switchover

This function triggers a switchover event identified by protection-group name.

Syntax

```
s_int32_t
oam_api_lps_switchover (u_int32_t vr_id, char *grp_name_str,
                        lps_switch_over switch_over)
```

Input Parameters

`vr_id`	VR instance ID
`grp_name_str`	The protection-group name
`switch_over`	Identifies a switchover event

Output Parameters

None

Return Values

OAM_API_SET_ERROR_PSC_GRP_LOOKUP when the protection-group lookup fails

OAM_API_SET_ERROR_PRIMARY_ME_NOT_SET when the primary ME is not configured for protection-group

OAM_API_SET_ERROR_BACKUP_ME_NOT_SET when the backup ME is not configured for protection-group

OAM_API_LOWER_PRIO_CMD_NOT_ALLOWED when a lower-priority event for switchover is triggered

OAM_API_MANUAL_SWITCH_ALREADY_EXISTS when manual switchover event is triggered which already exists

OAM_API_FORCE_SWITCH_ALREADY_EXISTS when force switchover event is triggered which already exists

OAM_API_SET_ERROR_MODE_SCHEME_MISMATCH when there is a mode or scheme mismatch

OAM_API_LPS_SET_ERROR when the function fails due to internal errors such as a memory failure

OAM_API_LPS_SET_SUCCESS when the function succeeds

oam_api_lps_wtr_timer_set

This function sets the WTR (wait-to-restore) timer value of the protection-group identified by protection-group name.

Syntax

```
s_int32_t
```

```
oam_api_lps_wtr_timer_set (u_int32_t vr_id, char *grp_name_str, u_int32_t timer)
```

Input Parameters

`vr_id`	VR instance ID
`grp_name_str`	The protection-group name
`timer`	Wait to respond timer value in seconds; input range is <0-720>

Output Parameters

None

Return Values

OAM_API_SET_ERR_INVALID_WTR_TIMER_VALUE when the WTR timer value is invalid

OAM_API_SET_ERROR_PSC_GRP_LOOKUP when the protection-group lookup fails

OAM_API_LPS_SET_SUCCESS when the function succeeds

© 2015 IP Infusion Inc. Proprietary

CHAPTER 22 MPLS-TP Ring Protection Switching API

The functions in this chapter are called by the RPS commands used to configure and manage protection switching coordination.

Functions

Function	Description
oam_api_rps_backup_meg_mip_set	Sets the backup MEG and ITU-T MIP for a protection group
oam_api_rps_backup_meg_mip_unset	Removes the backup MEG and ITU-T MIP for a protection group
oam_api_rps_clear_wtr_timer	Clears the WTR timer for a protection group
oam_api_rps_group_create	Creates a ring protection-group
oam_api_rps_group_delete	Deletes a ring protection-group
oam_api_rps_hold_off_timer_set	Sets a hold-off timer for protection events
oam_api_rps_primary_meg_mip_set	Sets the primary MEG and ITU-T MIP for a protection group
oam_api_rps_primary_meg_mip_unset	Removes the primary MEG and ITU-T MIP for a protection group
oam_api_rps_wtr_timer_set	Sets a wait-to-restore timer for a protection group

oam_api_rps_backup_meg_mip_set

This API sets the backup MEG and MP for the protection-group identified by the protection-group name.

Syntax

```
s_int32_t
oam_api_rps_backup_meg_mip_set (u_int32_t vr_id, char *grp_name_str,
                                char *meg_name, u_int16_t mp_id)Input Parameters
```

vr_id	VR instance ID
grp_name_str	Protection group
meg_name	Maintenance entity group name
mp_id	ITU-T MIP identifier <1-8191>

Output Parameters

None

Return Values

OAM_API_RPS_SET_ERR_GRP_LOOKUP when the protection group lookup fails

OAM_API_RPS_SET_ERR_INVALID_MEG_NAME when `meg_name` is not valid

OAM_API_RPS_SET_ERR_MEG_NAME_EXCEEDS_MAX_LEN when the `meg_name` exceeds the maximum length

OAM_API_RPS_ERR_MEG_ENTRY_NOT_EXISTS when `meg_name` does not exists

OAM_API_RPS_ERR_ITUT_OAM_ON_IETF_MEG when the MEG type is not ITU-T

OAM_API_RPS_ERR_INVALID_MP_ID when `mp_id` is not valid

OAM_API_RPS_ERR_MP_ENTRY_NOT_EXISTS when the `mp_id` lookup fails

OAM_API_RPS_ERR_CONFIG_ON_MEP when configured on a MEP.

OAM_API_RPS_ERR_MEG_SERVICE_TYPE_NOT_TUNNEL when the service type of the MEG is not tunnel

OAM_API_RPS_ERR_MEG_TYPE_MISMATCH when the protection group type is ITU-T

OAM_API_RPS_ERR_PG_ALREADY_HAS_PRIMARY_MP when the protection group already has an associated primary MEP

OAM_API_RPS_ERR_MP_ALREADY_ASSOCIATED when the MP is already configured as a primary MEG

OAM_API_RPS_ERR_PRI_REV_MEG_ENTRY_NOT_EXISTS when the primary MEG entry does not exist

OAM_API_RPS_ERR_PRI_REV_ME_ENTRY_NOT_EXISTS when the primary ME entry does not exist

OAM_API_RPS_ERR_BKP_REV_MEG_ENTRY_NOT_EXISTS when the backup MEG entry does not exist

OAM_API_RPS_ERR_BKP_REV_ME_ENTRY_NOT_EXISTS when the backup ME entry does not exist

OAM_API_RPS_SET_SUCCESS when the function succeeds

OAM_API_RPS_SET_ERROR when the function fails due to internal errors such as a memory failure

oam_api_rps_backup_meg_mip_unset

This function removes the backup MEG and MP for a protection group identified by the protection-group name.

Syntax

```
s_int32_t
oam_api_rps_backup_meg_mip_unset (u_int32_t vr_id, char *grp_name_str,
                                  char *meg_name, u_int16_t mp_id)Input Parameters
```

`vr_id`	VR instance ID
`grp_name_str`	Protection group
`meg_name`	Maintenance entity group name
`mp_id`	ITU-T MIP identifier <1-8191>

Output Parameters

None

Return Values

OAM_API_RPS_SET_ERR_GRP_LOOKUP when the protection-group lookup fails

OAM_API_RPS_ERR_ITUT_PG_DOES_NOT_EXIST when the protection group type is not ITU-T

OAM_API_RPS_ERR_MEG_ENTRY_NOT_EXISTS when the MEG entry does not exist

OAM_API_RPS_ERR_INVALID_MP_ID when `mp_id` is not valid

OAM_API_RPS_ERR_MP_ENTRY_NOT_EXISTS when the `mp_id` lookup fails

 © 2015 IP Infusion Inc. Proprietary

OAM_API_RPS_ERR_MP_ENTRY_NOT_IN_PG when the MP entry is not associated with the protection group

OAM_API_RPS_ERR_PRIMARY_NOT_CONFIGURED when a backup MEG is not configured for this protection group

OAM_API_RPS_ERR_INVALID_PRIMARY_MEG_ENTRY when `meg_name` is not valid

OAM_API_RPS_SET_SUCCESS when the function succeeds

OAM_API_RPS_SET_ERROR when there is an internal error

oam_api_rps_clear_wtr_timer

This function clears the WTR timer of the protection-group identified by the protection-group name.

Syntax

```
s_int32_t
oam_api_rps_clear_wtr_timer(u_int32_t vr_id, char *grp_name_str)
```

Input Parameters

`vr_id`	VR instance ID
`grp_name_str`	Protection group

Output Parameters

None

Return Values

OAM_API_RPS_SET_ERR_INVALID_WTR_TIMER_VALUE when the `wtr-timer` value is invalid

OAM_API_RPS_SET_ERR_GRP_LOOKUP when the protection-group lookup fails

RPS_FAILURE when there is an internal error

RPS_SUCCESS when the function succeeds

oam_api_rps_group_create

This function creates a ring protection-group based on the protection-group name or returns a ring protection-group configured with the same name.

Syntax

```
s_int32_t
oam_api_rps_group_create (u_int32_t vr_id, char *grp_name_str,
                          struct mpls_tp_rps_group **protection_grp)
```

Input Parameters

`vr_id`	VR instance ID
`grp_name_str`	Protection group

Output Parameters

`protection_grp`	Protection group

Return Values

OAM_API_RPS_SET_ERR_INVALID_VALUE when the value is invalid

OAM_API_RPS_SET_ERR_GRP_NAME_EXCEEDS_MAX_LEN when the protection group-name exceeds maximum length

OAM_API_RPS_SET_ERR_VR_NOT_EXIST when `vr-id` is not found

OAM_API_RPS_SET_ERROR when the function fails due to internal errors such as a memory failure

OAM_API_RPS_SET_ERR_GRP_CREATE when the function fails because of an internal error such as a memory failure

OAM_API_RPS_SET_ERR_GRP_INSERT when the function fails because of an internal error such as a insertion failure

OAM_API_RPS_SET_SUCCESS when the function succeeds

oam_api_rps_group_delete

This function deletes a ring protection-group based on the protection-group name.

Syntax

```
s_int32_t
oam_api_rps_group_delete (u_int32_t vr_id, char *grp_name_str)
```

Input Parameters

`vr_id`	VR instance ID
`grp_name_str`	Protection group

Output Parameters

None

© 2015 IP Infusion Inc. Proprietary

Return Values

OAM_API_RPS_SET_ERR_INVALID_VALUE when the value is invalid

OAM_API_RPS_SET_ERR_GRP_NAME_EXCEEDS_MAX_LEN when the protection group-name exceeds maximum length

OAM_API_RPS_SET_ERR_VR_NOT_EXIST when `vr-id` is not found

OAM_API_RPS_SET_SUCCESS when the function succeeds

OAM_API_RPS_SET_ERROR when the function fails due to internal errors such as a memory failure

OAM_API_RPS_SET_ERR_GRP_LOOKUP when the protection-group lookup fails

oam_api_rps_hold_off_timer_set

This function sets the hold_off timer value of the protection-group identified by the protection-group name.

Syntax

```
s_int32_t
oam_api_rps_hold_off_timer_set (u_int32_t vr_id, char *grp_name_str,
                                u_int32_t hold_off_timer)
```

Input Parameters

`vr_id`	VR instance ID
`grp_name_str`	Protection group
`hold_off_timer`	Timer value in seconds <0-720>

Output Parameters

None

Return Values

OAM_API_RPS_SET_ERR_INVALID_HOLD_OFF_TIMER_VALUE when the `hold-off-timer` value is invalid

OAM_API_RPS_SET_ERR_GRP_LOOKUP when the protection-group lookup fails

OAM_API_RPS_SET_SUCCESS when the function succeeds

oam_api_rps_primary_meg_mip_set

This function sets the primary MEG (Maintenance Entity Group) and ITU-T MIP (ME Intermediate Point) for the protection-group identified by the protection-group name.

Syntax

```
s_int32_t
oam_api_rps_primary_meg_mip_set (u_int32_t vr_id, char *grp_name_str,
                                 char *meg_name, u_int16_t mp_id)Input Parameters
```

`vr_id`	VR instance ID
`grp_name_str`	Protection group
`meg_name`	Maintenance entity group name
`mp_id`	ITU-T MIP identifier <1-8191>

Output Parameters

None

Return Values

OAM_API_RPS_SET_ERR_GRP_LOOKUP when the protection group lookup fails

OAM_API_RPS_SET_ERR_INVALID_MEG_NAME when `meg_name` is not valid

OAM_API_RPS_SET_ERR_MEG_NAME_EXCEEDS_MAX_LEN when the `meg_name` exceeds the maximum length

OAM_API_RPS_ERR_MEG_ENTRY_NOT_EXISTS when `meg_name` does not exists

OAM_API_RPS_ERR_ITUT_OAM_ON_IETF_MEG when the MEG type is not ITU-T

OAM_API_RPS_ERR_INVALID_MP_ID when `mp_id` is not valid

OAM_API_RPS_ERR_MP_ENTRY_NOT_EXISTS when the `mp_id` lookup fails

OAM_API_RPS_ERR_CONFIG_ON_MEP when configured on a MEP

OAM_API_RPS_ERR_MEG_SERVICE_TYPE_NOT_TUNNEL when the service type of the MEG is not tunnel

OAM_API_RPS_ERR_MEG_TYPE_MISMATCH when the protection group type is ITU-T

OAM_API_RPS_ERR_PG_ALREADY_HAS_PRIMARY_MP when the protection group already has an associated primary MEP

OAM_API_RPS_ERR_MP_ALREADY_ASSOCIATED when the MEP is already configured as a primary MEG

OAM_API_RPS_ERR_PRI_REV_MEG_ENTRY_NOT_EXISTS when the primary MEG entry does not exist

OAM_API_RPS_ERR_PRI_REV_ME_ENTRY_NOT_EXISTS when the primary ME entry does not exist

OAM_API_RPS_ERR_BKP_REV_MEG_ENTRY_NOT_EXISTS when the backup MEG entry does not exist

OAM_API_RPS_ERR_BKP_REV_ME_ENTRY_NOT_EXISTS when the backup ME entry does not exist

OAM_API_RPS_SET_SUCCESS when the function succeeds

OAM_API_RPS_SET_ERROR when the function fails due to internal errors such as a memory failure

oam_api_rps_primary_meg_mip_unset

This function removes the primary MEG and MP for the protection-group identified by the protection-group name.

Syntax

```
s_int32_t
oam_api_rps_primary_meg_mip_unset (u_int32_t vr_id, char *grp_name_str,
                                   char *meg_name, u_int16_t mp_id)Input Parameters
```

`vr_id`	VR instance ID
`grp_name_str`	Protection group
`meg_name`	Maintenance entity group name
`mp_id`	ITU-T MIP identifier <1-8191>

Output Parameters

None

© 2015 IP Infusion Inc. Proprietary

Return Values

OAM_API_RPS_SET_ERR_GRP_LOOKUP when the protection-group lookup fails

OAM_API_RPS_ERR_ITUT_PG_DOES_NOT_EXIST when the protection group type is not ITU-T

OAM_API_RPS_ERR_MEG_ENTRY_NOT_EXISTS when the MEG entry does not exist

OAM_API_RPS_ERR_INVALID_MP_ID when `mp_id` is not valid

OAM_API_RPS_ERR_MP_ENTRY_NOT_EXISTS when the `mp_id` lookup fails

OAM_API_RPS_ERR_MP_ENTRY_NOT_IN_PG when the MP entry is not associated with the protection group

OAM_API_RPS_ERR_PRIMARY_NOT_CONFIGURED when a primary MEG is not configured for this protection group

OAM_API_RPS_ERR_INVALID_PRIMARY_MEG_ENTRY when `meg_name` is not valid

OAM_API_RPS_SET_SUCCESS when the function succeeds

OAM_API_RPS_SET_ERROR when there is an internal error

oam_api_rps_wtr_timer_set

This function sets the wait-to-restore timer value of the protection-group identified by the protection-group name.

Syntax

```
s_int32_t
oam_api_rps_wtr_timer_set (u_int32_t vr_id, char *grp_name_str,
                           u_int32_t timer)
```

Input Parameters

`vr_id`	VR instance ID
`grp_name_str`	Protection group
`wtr_timer`	Timer in seconds <0-720>

Output Parameters

None

Return Values

OAM_API_RPS_SET_ERR_INVALID_WTR_TIMER_VALUE when the `wtr-timer` value is invalid

OAM_API_RPS_SET_ERR_GRP_LOOKUP when the protection-group lookup fails

OAM_API_RPS_SET_SUCCESS when the function succeeds

© 2015 IP Infusion Inc. Proprietary

CHAPTER 23 MPLS Non-stop Forwarding API

This chapter describes the MPLS-NSF (non-stop forwarding) feature for ZebOS-XP.

API

The following functions support MPLS-NSF. They perform checkpointing to synchronize the data from Active to Standby.

nsm_mpls_ftn_add_msg_process

This function is a handler for all FTM add messages.

Syntax

```
s_int32_t
nsm_mpls_ftn_add_msg_process (struct nsm_master *nm,
                              struct ftn_add_data *data,
                              struct ftn_ret_data *ret_data,
                              mpls_ftn_type_t f_type,
                              bool_t config)
```

Input Parameters

`*nm`	Pointer to the NSM master structure
`*data`	FTN add data
`*ret_data`	FTN return data
`f_type`	FTN type
`config`	Configuration

Output Parameters

None

Return Value

PAL_TRUE when the function succeeds

PAL_FALSE when the function fails

nsm_gmpls_ftn_del_msg_process

This function is a handler for FTN delete messages.

Syntax

```
s_int32_t
nsm_gmpls_ftn_del_msg_process (struct nsm_master *nm,
```

```
                                u_int32_t vrf_id,
                                struct fec_gen_entry *fec,
                                u_int32_t ftn_ix)
```

Input Parameters

`*nm`	Pointer to the NSM master structure
`vrf_id`	Virtual routing/forwarding instance ID
`*fec`	Forwarding equivalency class
`ftn_ix`	FTN table index

Output Parameters

None

Return Value

PAL_TRUE when the function succeeds

PAL_FALSE when the function fails

nsm_gmpls_ilm_del_msg_process

This function is a handler for ILM delete messages.

Syntax

```
s_int32_t
nsm_gmpls_ilm_del_msg_process(struct nsm_master *nm,
                              struct ilm_add_gen_data *data)
```

Input Parameters

`*nm`	Pointer to the NSM master structure
`data`	ILM generated data

Output Parameters

None

Return Value

PAL_TRUE when the function succeeds

PAL_FALSE when the function fails

nsm_mpls_rib_if_up_process

This function is called when an interface comes up.

Syntax

```
void
nsm_mpls_rib_if_up_process (struct interface *ifp)
```


© 2015 IP Infusion Inc. Proprietary

Input Parameters

`ifp`	Pointer to the interface

Output Parameters

None

Return Value

PAL_TRUE when the function succeeds

PAL_FALSE when the function fails

nsm_mpls_rib_if_down_process

This function is called when an interface goes down.

Syntax

```
void
nsm_mpls_rib_if_down_process (struct interface *ifp)
```

Input Parameters

`ifp`	Pointer to the interface

Output Parameters

None

Return Value

PAL_TRUE when the function succeeds

PAL_FALSE when the function fails

© 2015 IP Infusion Inc. Proprietary

CHAPTER 24 Virtual Circuit Interface API

This chapter contains an overview of the Virtual Circuit Module API. For details about the functions, see the *Multi-Protocol Label Switching Forwarder Developer Guide.*

Virtual Circuit Interface API

The MPLS Forwarder package provides a static library (libmpls_client.a) that can be linked to a user-space process to populate Virtual Circuit related FTN and ILM entries in the MPLS Forwarder.

APIs	Description
ipi_mpls_vc_end	This function unbinds a specific interface from the virtual circuit it was bound to.
ipi_mpls_vc_ftn_entry_add	This function adds a Virtual Circuit FTN entry to the incoming interface that is bound to a virtual circuit. The addition of an FTN entry requires you to name the protocol adding it, as well as the opcode that the MPLS Forwarder should use when treating a frame received on the incoming interface. The possible opcodes are PUSH_FOR_VC and PUSH_AND_LOOKUP_FOR_VC. This routine can also be used to modify an existing FTN entry.
ipi_mpls_vc_ftn_entry_del	This function removes a specific Virtual Circuit FTN entry from a designated incoming interface. When an entry is not present, a warning is logged, and the request is ignored. If the protocol requested does not match the one stored in the FTN entry on the interface, the delete operation fails.
ipi_mpls_vc_init	This function binds a specific virtual circuit to a particular interface. An interface may be bound to only one virtual circuit and a virtual circuit also can be bound to only one interface. Once an interface is tied to a virtual circuit, all the packets received on that interface are tunneled over the corresponding virtual circuit. If there is no Virtual Circuit FTN entry associated with an interface, the MPLS Forwarder drop all packets on the interface.

MPLS Virtual Circuit Opcodes

The following MPLS opcodes are part the MPLS forwarder and handle packets transported on virtual circuits.

OPCODES	Description
PUSH_AND_LOOKUP_FOR_VC	Push a VC label to encapsulate an incoming Layer 2 PDU and lookup the MPLS FTN table to get an LSP to reach the tunnel end-point. This opcode is used for non-directly-connected VC PE routers
PUSH_FOR_VC	Push a VC label to encapsulate an incoming Layer 2 PDU. This opcode is used when the VC PE routers are adjacent
POP_FOR_VC	POP the specified label from the labeled packet and transmit the de-encapsulated Layer-2 PDU on the outgoing interface. This opcode is used for VC labels only.

© 2015 IP Infusion Inc. Proprietary

CHAPTER 25 Differentiated Services API

This chapter describes the Differentiated Services API for ZebOS-XP which is based on RFC 3270, MPLS Support of Differentiated Services.

API

This section contains the functions in the Differentiated Services API.

Function	Description
nsm_mpls_dscp_exp_map_add	Adds a class-to-EXP bit mapping
nsm_mpls_dscp_exp_map_del	Deletes a class-to-EXP bit mapping
nsm_mpls_dscp_support_add	Adds support for a Diffserv class
nsm_mpls_dscp_support_del	Deletes support for a Diffserv class

Include File

To call the functions in this chapter, you must include `nsm/mpls/nsm_mpls.h`.

nsm_mpls_dscp_exp_map_add

This function adds a class-to-EXP bit mapping. This function implements the `mpls class-to-exp-bit` command.

Syntax

```
int
nsm_mpls_dscp_exp_map_add (struct nsm_master *nm, char *name, int exp_val);
```

Input Parameters

`nm`	A pointer to the NSM master
`name`	Class alias
`exp_val`	EXP bits

Output Parameters

None

Return Value

NSM_SUCCESS when the function succeeds

NSM_FAILURE when MPLS is not found

NSM_ERR_DSCP_INVALID when the class is invalid

NSM_ERR_DSCP_NOT_SUPPORTED when the class is unsupported

nsm_mpls_dscp_exp_map_del

This function deletes a class-to-EXP bit mapping. This function implements the `no mpls class-to-exp-bit` command.

Syntax

```
int
nsm_mpls_dscp_exp_map_del (struct nsm_master *nm, char *name, int exp_val);
```

Input Parameters

`nm`	A pointer to the NSM master
`name`	Class alias
`exp_val`	EXP bits

Output Parameters

None

Return Value

NSM_SUCCESS when the function succeeds

NSM_FAILURE when MPLS is not found

NSM_ERR_DSCP_INVALID when the class is invalid

NSM_ERR_DSCP_EXP_MISMATCH when the given EXP bits do not match the EXP bits in the list

nsm_mpls_dscp_support_add

This function adds support for a Diffserv class. This function implements the `mpls support-diffserv-class` command.

Syntax

```
intnsm_mpls_dscp_support_add (struct nsm_master *nm, char *name);
```

Input Parameters

`nm`	A pointer to the NSM master
`name`	Class alias

Output Parameters

None

Return Value

NSM_SUCCESS when the function succeeds

NSM_ERR_DSCP_INVALID when the class is invalid

NSM_FAILURE when MPLS is not found

 © 2015 IP Infusion Inc. Proprietary

nsm_mpls_dscp_support_del

This function deletes support for a Diffserv class. This function implements the `no mpls support-diffserv-class` command.

Syntax

```
int
nsm_mpls_dscp_support_del (struct nsm_master *nm, char *name);
```

Input Parameters

`nm`	A pointer to the NSM master
`name`	Class alias

Return Value

NSM_SUCCESS when the function succeeds

NSM_ERR_DSCP_INVALID when the class is invalid

NSM_FAILURE when MPLS is not found

© 2015 IP Infusion Inc. Proprietary

Appendix A MPLS Messages

This appendix describes MPLS messages.

MPLS Messages

Message Constant	Source
NSM_MSG_MPLS_VC_ADD	Sent from NSM to protocol whenever an L2 Virtual Circuit is added to the system
NSM_MSG_MPLS_VC_DELETE	Sent from NSM to protocol whenever an L2 Virtual circuit is deleted from the system
NSM_MSG_MPLS_FTN_IPV4	Sent from protocol to NSM to add, delete, or slow-delete an IPv4 FTN entry
NSM_MSG_MPLS_FTN_IPV6	Sent from protocol to NSM to add or delete an IPv6 FTN entry
NSM_MSG_MPLS_FTN_REPLY	Sent from NSM to protocol to add, delete or slow-delete an FTN entry
NSM_MSG_MPLS_ILM_IPV4	Sent from protocol to NSM to add or delete an IPv4 ILM entry
NSM_MSG_MPLS_ILM_IPV6	Sent from protocol to NSM to add or delete an IPv6 ILM entry
NSM_MSG_MPLS_ILM_REPLY	Sent from NSM to protocol to add or delete an ILM entry
NSM_MSG_MPLS_VC_FTN_IPV4	Sent from protocol to NSM to add or delete an IPv4 FTN entry for a Virtual Circuit
NSM_MSG_MPLS_VC_FTN_IPV6	Sent from protocol to NSM to add or delete an IPv6 FTN entry for a Virtual Circuit
NSM_MSG_MPLS_VC_ILM_IPV4	Sent from protocol to NSM to add or delete an IPv4 ILM entry for a Virtual Circuit
NSM_MSG_MPLS_VC_ILM_IPV6	Sent from protocol to NSM to add or delete an IPv6 ILM entry for a Virtual Circuit
NSM_MSG_MPLS_VC_FTN_REPLY	Sent from NSM to protocol in response to a request to add or delete an FTN entry
NSM_MSG_MPLS_VC_ILM_REPLY	Sent from NSM to protocol in response to a request to add or delete an ILM entry
NSM_MSG_MPLS_NOTIFICATION	Sent from NSM to protocol for notifications
NSM_MSG_MPLS_BIDIR_FTN_REPLY	NSM sends to protocol about FTN bidir reply message
NSM_MSG_MPLS_BIDIR_ILM_REPLY	NSM sends to protocol about ILM bidir reply message
NSM_MSG_MPLS_ECHO_REQUEST	Protocol sends echo request to NSM to check the connectivity status

Message Constant	Source
NSM_MSG_MPLS_FTN_ADD_REPLY	NSM sends to protocol about the FTN add reply
NSM_MSG_MPLS_FTN_DEL_REPLY	NSM sends to protocol about the FTN delete reply
NSM_MSG_MPLS_FTN_DOWN	NSM send the FTN Down event to the protocol when BFD session for the FEC goes down
NSM_MSG_MPLS_FTN_GEN_IPV4	Protocol send about the MPLS FTN add/delete information to NSM
NSM_MSG_MPLS_FTN_GEN_REPLY	Protocol module receives general FTN reply from NSM.
NSM_MSG_MPLS_IGP_SHORTCUT_LSP	NSM sends IGP shortcut lsp message to protocol
NSM_MSG_MPLS_IGP_SHORTCUT_ROUTE	NSM receives IGP shortcut route updates from protocol.
NSM_MSG_MPLS_ILM_ADD_REPLY	NSM send ILM add reply to the protocol
NSM_MSG_MPLS_ILM_DEL_REPLY	NSM send ILM delete reply to the protocol.
NSM_MSG_MPLS_ILM_GEN_ADD_REPLY	ILM reply receives general add reply for the uni-directional and bi-directional add from the NSM
NSM_MSG_MPLS_ILM_GEN_DEL_REPLY	ILM reply receives general delete reply for the uni-directional and bi-directional add from the NSM
NSM_MSG_MPLS_ILM_GEN_IPV4	Protocol send about the ILM add/delete information to NSM
NSM_MSG_MPLS_ILM_GEN_REPLY	NSM send ILM gen reply message to the protocol
NSM_MSG_MPLS_MS_PW	NSM sends the MSPW add/del/update message to the protocols that have subscribed for the MPLS VC service
NSM_MSG_MPLS_OAM_ERROR	NSM sends protocols if the msg type is failed
NSM_MSG_MPLS_OAM_ITUT_PROCESS_REQ	Protocol sends OAM process request to NSM
NSM_MSG_MPLS_OAM_L3VPN	Protocol send VPN rd details to NSM
NSM_MSG_MPLS_PING_REPLY	NSM sends the echo reply message to the protocol
NSM_MSG_MPLS_PW_STATUS	NSM sends the pw status message to the protocol
NSM_MSG_MPLS_TRACERT_REPLY	NSM sends the exact instance that caused this reply to be sent to the protocol
NSM_MSG_MPLS_VC_FIB_ADD	NSM sends the VC FIB add message to the protocol
NSM_MSG_MPLS_VC_FIB_DELETE	NSM sends the VC FIB delete message to the protocol
NSM_MSG_MPLS_VC_FTN_ADD_REPLY	Unused
NSM_MSG_MPLS_VC_FTN_DEL_REPLY	Unused
NSM_MSG_MPLS_VC_ILM_ADD_REPLY	Unused

© 2015 IP Infusion Inc. Proprietary

Message Constant	Source
NSM_MSG_MPLS_VC_ILM_DEL_REPLY	Unused
NSM_MSG_MPLS_VC_TUNNEL_INFO	Protocols send the VC tunnel information to NSM
NSM_MSG_VC_ILM_ADD	Unused
NSM_MSG_STALE_INFO	NSM sends to LDP to free stale FEC information

MPLS-TP

Message Constant	Source
NSM_MSG_MPLS_TP_MEG_STATUS_CHANGE	NSM sends to MPLS TP OAMD when the status of a MEG changes
NSM_MSG_MPLS_TP_PATH_INFO_QUERY	Client sends to NSM to request transport path information (tunnel or virtual circuit)
NSM_MSG_MPLS_TP_PATH_INFO_QUERY_RESP	NSM sends this message to NSM client in response to NSM_MSG_MPLS_TP_PATH_INFO_QUERY
NSM_MSG_GLOBAL_ID_NODE_ID_UPDATE	NSM sends to client about the MPLS TP global identifier and node identifier

VPLS Messages

Message Constant	Source
NSM_MSG_VPLS_ADD	Sent from NSM to LDP to add or update VPLS instances if at least one mesh peer is defined for a VPLS instance
NSM_MSG_VPLS_DELETE	Sent from NSM to LDP to delete VPLS instance when either a VPLS instance is deleted or the entire VPLS mesh peer configuration is removed
NSM_MSG_VPLS_FIB_ADD	Sent from LDP to NSM to add FIB entries related to spoke/mesh VC for a VPLS instance
NSM_MSG_VPLS_FIB_DELETE	Sent from LDP to NSM to delete FIB entries related to spoke/mesh VC for a VPLS instance
NSM_MSG_VPLS_MAC_WITHDRAW	Sent from NSM to LDP to trigger generation of MAC withdrawal message from LDP to its core VPLS peers for a VPLS instance; also sent from LDP to NSM to delete the specified MAC addresses plane for the VPLS instance

© 2015 IP Infusion Inc. Proprietary

© 2015 IP Infusion Inc. Proprietary

Appendix B Glossary

Note: This is not an exhaustive glossary of the terms used throughout this manual.

Term	Definition
ACH	Associated Channel Header
AGI	Attachment Group Identifier
AII	Attachment Individual Identifier
AIS	Alarm Indicator Signaling
API	Application Program Interface
AS	Autonomous System
BFD	Bidirectional Forwarding Detection
BW	Bandwidth
CC	Continuity Check
CC-Type	Control Channel Type
CE	Customer Edge (router)
CLI	Command Line Interface
CV-Type	Connectivity Verification Type
CW	Control Word
DM	Delay Measurement
FEC	Forwarding Equivalence Class
FIB	Forwarding Information Base
FM	Fault Management
FTN	FEC to NHLFE
G-ACh	Generic Associated Channel
ILM	Incoming Label Map
IMI	Interactive Management Interface
IMISH	Interactive Management Interface Shell
LB	Loopback
LDI	Link Down Indication

Term	Definition
LDP	Label Distribution Protocol
LI	Lock Instruct
LKR	Lock Report
LSE	Label Stack Entry
LSP	Label Switched Path
LSR	Label Switched Router
ME	Maintenance Entity
MEG	Maintenance Entity Group
MEP	MEG End Point
MIP	MEG Intermediate Point
MPLS	Multi-Protocol Label Switching
MPLS TP	MPLS Transport Profile
MS-PW	Multi-Segment Pseudo wires
NHLFE	Next Hop Label Forwarding Entry
NSM	Network Service Module
OAM	Operations, Administration and Maintenance
OAMD	OAM Daemon
P2MP	Point-to-Multipoint
P2P	Point-to-Point
u-PE	A Layer 2 device used for Layer 2 aggregation
PE	Provider’s Edge (router)
PM	Protocol Module
PTP	Precision Time Protocol
PW	Pseudo Wire
RIB	Routing Information Base
RSVP	Resource-Reservation Protocol
SAII	Source Attachment Individual Identifier
TAII	Target Attachment Individual Identifier
TC	Traffic Class
TLV	Type, Length and Value

© 2015 IP Infusion Inc. Proprietary

Term	Definition
UDP	User Datagram Protocol
VC	Virtual Circuit
VCCV	Virtual Circuit Connectivity Verification
VE	A VPLS Edge Device
VPLS	Virtual Private LAN Service
XC	Cross-connect

© 2015 IP Infusion Inc. Proprietary

Index

L

M

N

© 2015 IP Infusion Inc. Proprietary

O

P

R

S

T

© 2015 IP Infusion Inc. Proprietary

U

V

 © 2015 IP Infusion Inc. Proprietary