ZebOS-XP®
Network Platform

Version 1.4
Extended Performance

Multi-Protocol Label Switching Forwarder
Developer Guide

December 2015

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

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

Contents

Preface

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

Audience

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

Conventions

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

Table P-1: Conventions

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

Contents

This document contains these chapters and appendices:

- Chapter 1, *ZebOS-XP MPLS Forwarder*
- Chapter 2, *Data Structures*
- Chapter 3, *MPLS Data Link Module*
- Chapter 4, *FTN and VRF Label Interface APIs*

Related Documents

The following guides are related to this document:

- *Multi-Protocol Label Switching Command Reference*
- *Multi-Protocol Label Switching Configuration Guide*
- *Multi-Protocol Label Switching Developer Guide*
- *Network Services Module Command Reference*
- *Network Services Module Developer Guide*

- *Installation Guide*
- *Architecture Guide*

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

Support

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

Comments

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

CHAPTER 1 ZebOS-XP MPLS Forwarder

MPLS is a label swapping and forwarding technology in which every packet contains a short, fixed-length label. Routers use these label values to forward incoming packets to their destination.

MPLS divides its functions into two distinct categories:

- Assigning and exchanging labels between Label Switching Routers (LSRs) through the Label Distribution Protocol (LDP).
- Forwarding labeled packets. The MPLS Forwarder swaps incoming labels with outgoing labels and forwards the resulting packets to the outgoing interface. MPLS supports the Ethernet interface.

Internal Architecture

The MPLS Forwarder processes incoming packets from all the network interfaces. Its primary function is to forward traffic over data paths created by label distribution protocols such as LDP and RSVP-TE.

The MPLS Forwarder handles the following functions:

- Receiving IP/MPLS unicast packets from the interface queues.
- Receiving Layer-2 packets from the interface queues.
- Forwarding labeled/unlabeled unicast packets based on IP address or MPLS labels.
- Forwarding labeled/unlabeled Layer-2 frames based on incoming interface or MPLS labels.
- Fragmenting packets exceeding the MTU size of the outgoing interface.

The MPLS Forwarder is made up of the following separate entities:

- A global FTN (FEC to Next-Hop-Label-Forwarding-Entry) table. The kernel interfaces use this table when processing non-labeled packets (IP packets) and when the kernel interface is not bound to a VRF table. Multiple kernel interfaces may use one global FTN table; or each interface may use its own table.
- One or more ILM (Incoming Label Map) tables. The kernel interfaces use these tables to process labeled packets and the label space in the interface contains a positive integer. An ILM table is created per label-space used. Therefore, if all interfaces in the system are using the same label-space, only one ILM table is created.
- One or more VRF tables. The kernel interfaces use these tables to process non-labeled packets and the interface is bound to a VRF table. Many kernel interfaces may use one VRF table; or each interface may use its own table.
- One or more interfaces with the flexibility to be enabled for either MPLS or VRF, or both.

If an interface is not enabled for MPLS or VRF, all labeled packets are dropped.

Operational Codes

The MPLS Forwarder supports the following operational code (opcodes) that designate the handling of labeled and unlabeled packets:

PUSH

Upon finding this opcode in an FTN entry, the MPLS Forwarder creates an MPLS shim on top of the IP packet, and PUSHes the specified label. When creating the Ethernet frame, the forwarder sets the frame type to ETH_P_MPLS_UC for a uni-cast MPLS packet.

POP

Upon finding this opcode in an ILM entry, the MPLS Forwarder POPs the specified label from the shim of the labeled packet. If this label was the last label in the stack, the packet is forwarded over IP.

SWAP

Upon finding this opcode in an ILM entry, the MPLS Forwarder SWAPs the specified label with a new outgoing label, and forwards the labeled packet accordingly.

POP_FOR_VPN

Upon finding this opcode in an ILM entry, the MPLS Forwarder POPs the specified label from the shim of the labeled packet. This label is the last one in the stack, and the ensuing unlabeled packet is forwarded based on the destination (next-hop and/or out-going interface) specified in the ILM entry.

PUSH_AND_LOOKUP

Upon finding this opcode in a VRF entry, the MPLS Forwarder creates an MPLS shim on top of the IP packet, and PUSHes the specified label. It then tries to lookup an FTN entry for the FEC in question in the global FTN table. If no FTN entry is found, the packet is dropped. When creating the Ethernet frame, the forwarder sets the frame type to ETH_P_MPLS_UC for a uni-cast MPLS packet.

DLVR_TO_IP

Upon finding this opcode in an FTN entry, the MPLS Forwarder forwards the IP packet based on the destination (next-hop and/or out-going interface) specified in the FTN entry.

PUSH_AND_LOOKUP_FOR_VC

Upon finding this opcode in an FTN entry, the MPLS Forwarder creates an MPLS shim on top of the Ethernet frame, and PUSHes the specified label. It then tries to lookup an FTN entry for the Virtual Circuit end-point specified in the global FTN table. If no FTN entry is found, the packet is dropped. If successful, the modified frame is encapsulated inside an Ethernet frame and forwarded.

PUSH_FOR_VC

Upon finding this opcode in an FTN entry, the MPLS Forwarder creates an MPLS shim on top of the Ethernet frame, and PUSHes the specified label. The modified frame is then encapsulated inside an Ethernet frame and forwarded.

POP_FOR_VC

Upon finding this opcode in an ILM entry, the MPLS forwarder POPs the specified label from the labeled packet. The resulting Layer-2 PDU is transmitted on the outgoing interface specified by the ILM entry.

SWAP_AND_LOOKUP

This opcode is added for the ILM entries. It is used for pushing an additional label onto the incoming packet.

Valid Label Ranges

In the scheme of labels, there are several reserved labels and reserved label ranges. For the detailed explanation of these see RFC 3032 section 2.1.

The range 0-3 is reserved for the following uses:

- 0 is the IPv4 Explicit NULL Label. When the Forwarder receives a packet with this label value at the bottom of the label stack, the stack pops the label value; it bases all forwarding of this packet on the IPv4 header.
- 1 is the Router Alert Label. When the Forwarder receives a packet with this label value at the top of the label stack, the forwarder uses the next label beneath it in the stack. The Router Alert Label is pushed on top of the stack if further forwarding is required.
- 2 is the IPv6 Explicit NULL Label. When the Forwarder receives a packet with this label value at the bottom of the label stack, it pops the label stack and forwards the packet based on the IPv6 header.
- 3 is the Implicit NULL Label. When the Forwarder receives a packet with this label value at the top of the label stack, the LSR pops the stack.

The range 4-15 is reserved for future use.

Manual Configuration for MPLS Label Stacking

In hierarchical label stacking, an MPLS packet contains more than one label but only the outermost (top) label is used for switching packets in the MPLS core. The inner labels may be used for forwarding when the outer label is popped at an LSR. This feature enables a hierarchy of LSPs where an LSP may be piggybacked on another LSP and is widely used for layer 2 and layer 3 MPLS based VPN solutions. A manual control option is provided to tunnel an incoming transit LSP over a locally configured LSP.

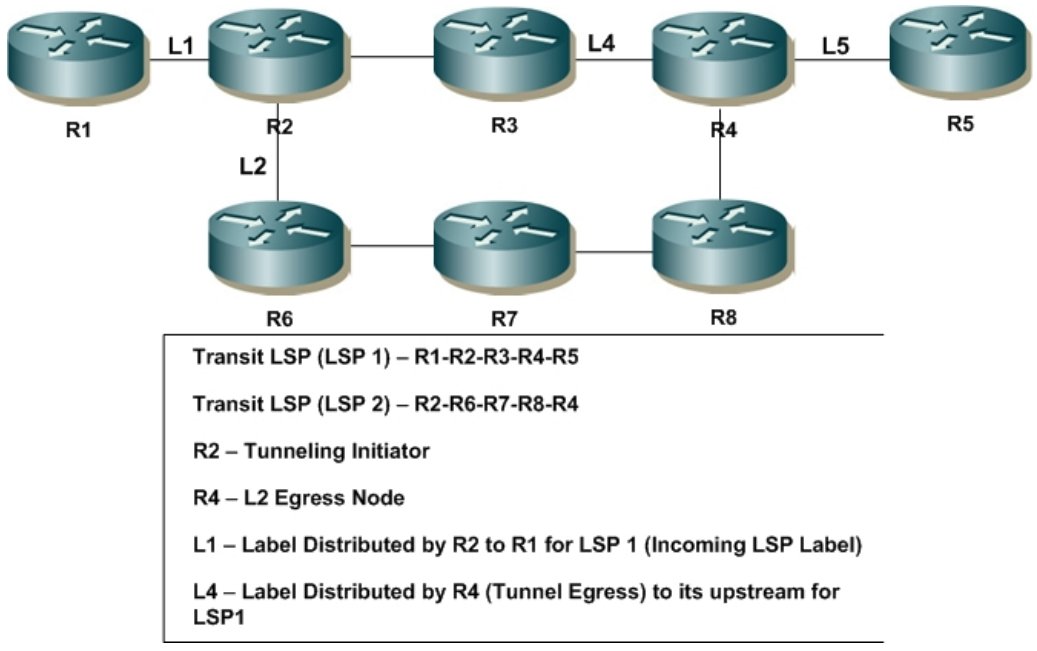

Figure 1-1: Label Stacking

In the diagram above, LSP1 can be tunneled over LSP2 by using a manual configuration on node R2, tunnel LSP2 ingress. The user needs to provide the incoming label (L1) for LSP1 and label L4 which will be used for forwarding LSP2 data after the tunneling operation is terminated at R4. In this scenario, router R4 should be able to accept L4 from any of the interfaces. This is achieved by using platform wide label spaces on R4.

When tunneling is configured on R2, all MPLS packets received with label L1 are swapped with label L4 and another label (L21) is pushed on top. The packet ID is forwarded to node R6. The additional label is required to tunnel the packet over LSP2. The forwarding engine at nodes R6, R7 and R8 only looks at the top label (LSP2 label) to forward packets to the tunnel egress (R4). At node R4, the top label is popped and label L4 (belonging to LSP1) is used to forward traffic to R5 over LSP1.

Virtual Private LAN Services (VPLS)

Virtual Private LAN Services (VPLS) is a way to provide ethernet based multipoint to multipoint communication over IP/MPLS networks. It allows geographically dispersed sites to share an ethernet broadcast domain by connecting sites through pseudo-wires. It uses a set of Martini circuits grouped by a common VPLS identifier to achieve this service objective. VPLS supports topologies like peer and spoke, where peer consists of full mesh VPLS pseudo wires in MPLS core and spokes consist of L3 tunnels connecting to VPLS LER's (Peer).

MPLS forwarder has been extended to support the below VPLS features:

- Traffic forwarding over a full VPLS mesh topology.
- Traffic forwarding over H-VPLS topology.
- MAC learning and ageing functionality.

System Architecture

VPLS Mesh Network

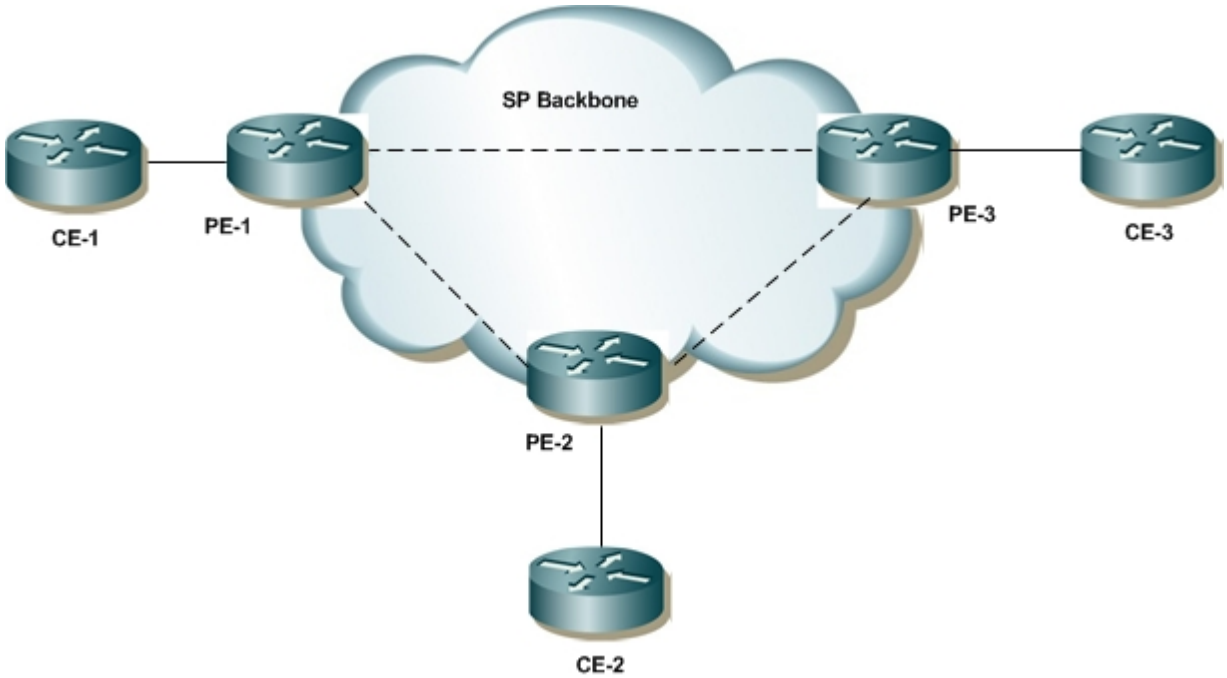

Figure 1-2: VPLS Mesh Topology

VPLS mesh helps to emulate a LAN service over VCs among multiple CEs by connecting to PEs. VPLS instance of unique VPN ID manages the VCs created for multiple neighbours and MAC learning for each neighbour. Each CE is bound to one PE with an attachment circuit.

Hierarchical-VPLS Network

H-VPLS reduces signalling and replication overhead which is present in VPLS mesh over a large scale of network. Full mesh connectivity between all the participating PEs is not required in case of H-VPLS.

A Spoke VC connection with one of the PEs in VPLS mesh is sufficient to extend a VPLS network as a hierarchical VPLS.

Spoke VC connection can be setup as any of the following types

- Non-bridging capable Spoke VC
- Bridging capable Spoke VC

Non-bridging capable Spoke VC

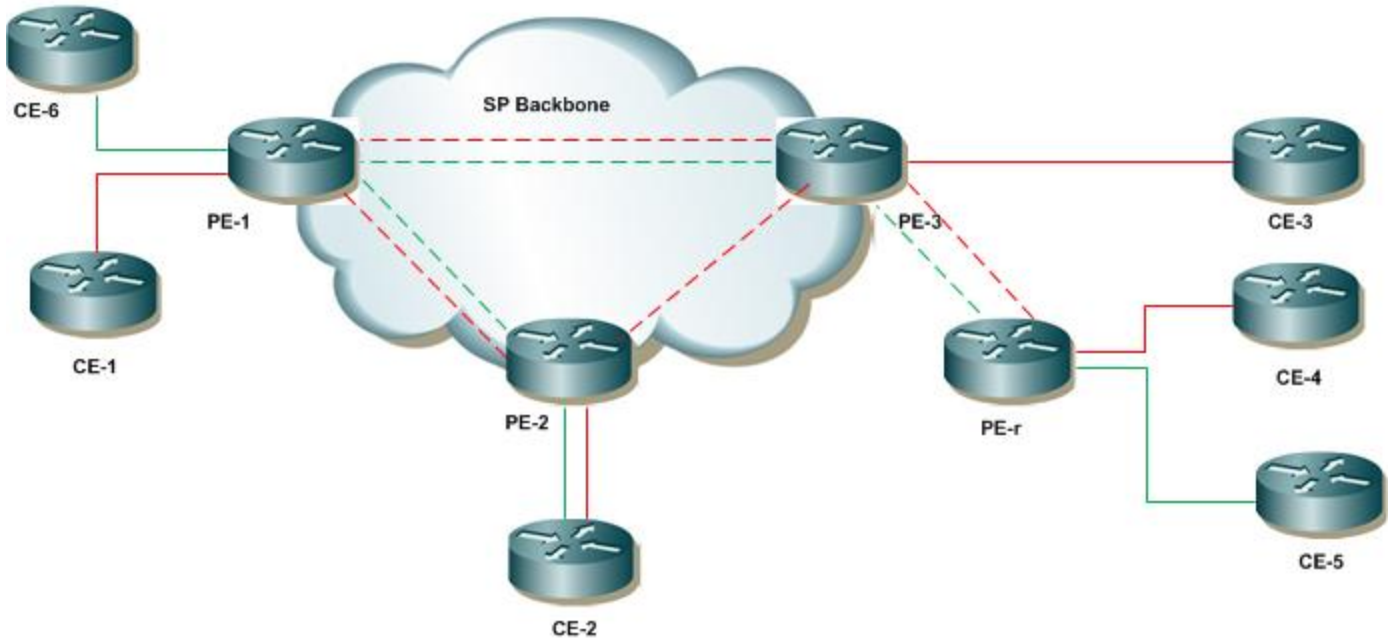

Figure 1-3: VPLS Non-bridging Topology

Non-bridging capable Spoke VC terminates at MPLS L2 circuit and AC. Individual Spoke VC connection is mandatory between PE3 and PE-r to serve each CE. Traffic is forwarded between Spoke VC and its associated AC. No mac learning happens at CE side as it's a MPLS L2 circuit at this end of Spoke VC.

Bridging capable Spoke VC

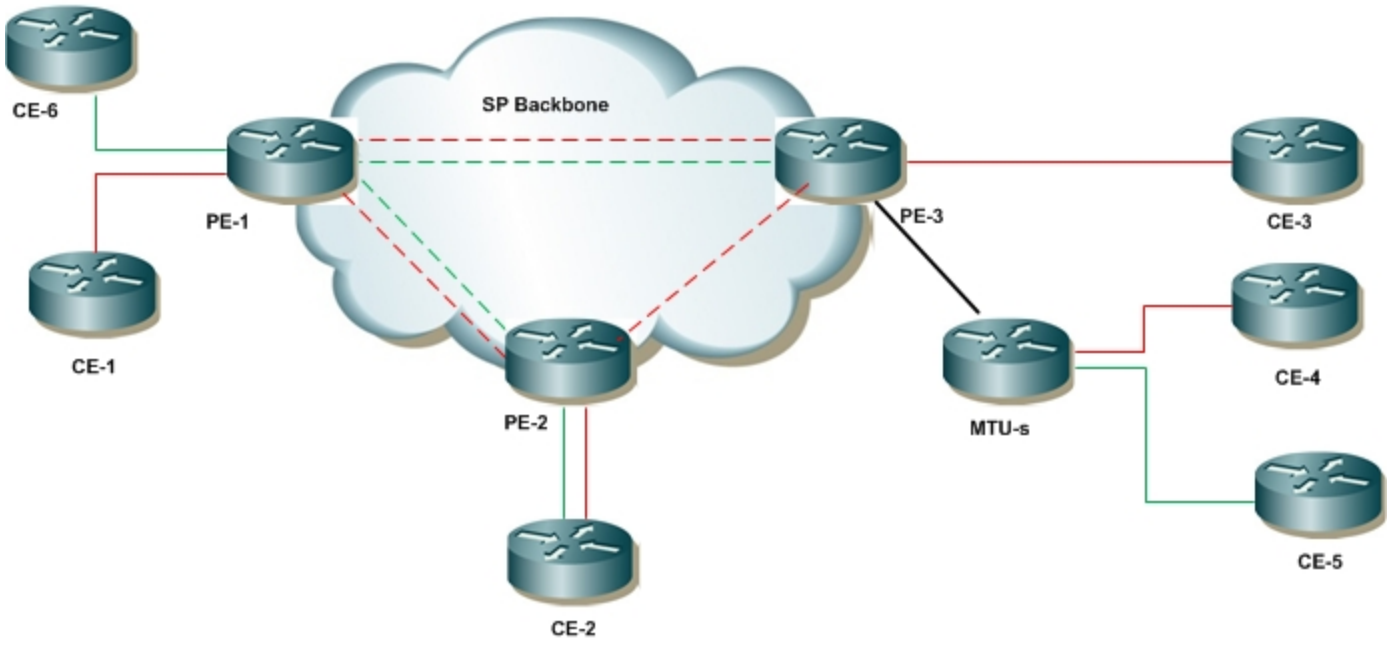

Figure 1-4: VPLS Bridging Topology

Bridging capable Spoke VC terminates at a VSI which can be bound to multiple ACs. Each AC in turn bounds to a CE. Only a single Spoke VC is sufficient between PE3 and MTU-s to serve multiple CEs.

Mac learning happens at CE side of MTU-s, as it's a VPLS instance which is bound to multiple ACs at this end of Spoke VC. Traffic is forwarded based on the MAC learnt on the ACs

When a PE router receives a packet on an attachment circuit, it sends it over all the VCs which are part of this VPLS instance. On receiving this broadcasted packet, each PE learns the MAC for the sender PE and it's updated in MAC table against VPN ID. Learnt MAC addressed are removed from table once they are age out and added to table when receives packet on specific ports.

CHAPTER 2 Data Structures

This chapter describes the data structures used by the MPLS Forwarder.

Common Data Structures

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

- `lib_globals`
- `pal_in4_addr`
- `pal_in6_addr`

if_ident

This struct identifies the interface.

Definition

```
struct if_ident
{
  u_int32_t if_index;
  char if_name[INTERFACE_NAMSIZ + 1];
};
```

ds_info_fwd

This struct contains DiffServ information.

Definition

```
struct ds_info_fwd
{
  u_char lsp_type;

  /* DSCP-to-EXP mapping for ELSP. */
  u_char dscp_exp_map[8];

  /* DSCP value for LLSP. */
  u_char dscp;

  /* AF set for LLSP. */
  u_char af_set;
};
```

```
#endif /* HAVE_DIFFSERV */

/* Key used by RSVP-TE protocol for IPV4 */
struct rsvp_key_ipv4_fwd
{
  u_int16_t trunk_id;
  u_int16_t lsp_id;
  struct pal_in4_addr ingr;
  struct pal_in4_addr egr;
};
```

mpls_owner_fwd

This struct contains MPLS owner information.

Definition

```
struct mpls_owner_fwd
{
  /* IPI_PROTO_xxx */
  u_char protocol;
  union
  {
    struct rsvp_key_fwd r_key;
  } u;
};
```

mpls_nh_fwd

This struct contains MPLS nexthop information.

Definition

```
struct mpls_nh_fwd
{
  u_char afi;
  union
  {
    u_char key;
    struct pal_in4_addr ipv4;
#ifdef HAVE_IPV6
    struct pal_in6_addr ipv6;
#endif
  } u;
};
```

CHAPTER 3 MPLS Data Link Module

The MPLS Data Link module processes all frames of type IP and MPLS. MPLS is supported on Ethernet (Linux platform, 2.4.x).

Ethernet Interface

Upon receipt of a frame by the Data Link device driver, the frame is passed to the MPLS module, which handles the processing of all packets of type IP or MPLS. The following steps are then taken:

1. Determine if packet is labeled: if the incoming packet is labeled, this is an MPLS packet; go to 2. Otherwise, the packet is an IP packet; go to 3
2. Use the top label in the packet to look up the destination in ILM table that this interface is bound to. If the lookup finds an outgoing label, go to 4. Else, drop the packet and exit this function.
3. By employing the best-match principle, use the destination IP address to determine the FEC that this destination address belongs to. Using this FEC as the key, lookup in the FTN table for a valid outgoing label. If we have a valid label for the destination address, push the outgoing label found in the FTN table onto the packet; continue with 6. Otherwise go to 5.
4. If there is no outgoing label for the incoming label, the LSR is an egress for the current LSP, and the packet shall be routed using traditional, native routing; continue with 5. Otherwise, push the mapped label onto the packet; continue with 8.
5. If the opcode associated with this ILM entry was POP_FOR_VC, go to 6. Else go to 7.
6. Remove the MPLS shim and pass the Ethernet frame that had been encapsulated with the MPLS shim to the outgoing interface's controller.
7. Decrement the TTL fields of the labeled or unlabeled IP packet, use the kernel URF to route the packet using conventional routing, then exit this function.
8. Decrement the TTL fields of the packet and label-switch the packet; then exit this function.

The following steps are taken in an environment where LDP, BGP and the MPLS Forwarder all exist together:

1. Determine if packet is labeled: if the incoming packet is labeled, this is an MPLS packet; go to 2. Otherwise, the packet is an IP packet; go to 3
2. Determine which ILM table to use to look up outgoing labels: if the interface that accepts incoming packets is bound to an ILM table (meaning that the label-space-value is not zero), use that ILM table. Else, because no ILM table is bound to this interface, use the global ILM table.Use the top label in the packet as a key to lookup an outgoing label in either the bound or global ILM table.If the lookup finds an outgoing label, go to 4. Else, drop the packet and exit this function.
3. By employing the best-match principle, use the destination IP address to figure out the FEC that this destination address belongs to. Using this FEC as the key, lookup in the FTN table for a valid outgoing label. The FTN table to be used is decided as follows: If there is a VRF table bound to this interface, lookup in that VRF table. If not, lookup in the global FTN table. If we have a valid label for the destination address, push the outgoing label found in the FTN table onto the packet; continue with 6. Otherwise go to 5.

4. If there is no outgoing label for the incoming label, the LSR is an egress for the current LSP, and the packet shall be routed using traditional native routing; continue with 5. Otherwise push the mapped label onto the packet; continue with 6.

5. If the lookup was done in a VRF table, drop the packet and exit this function. Otherwise, decrement the TTL fields of packet and use the kernel URF to route the packet using conventional routing. Exit this function.

6. Decrement the TTL fields of the and label-switch the packet. Exit this function.

The following steps are taken in an environment where LDP is being used to set up a Layer 2 Virtual Circuit between remote nodes:

1. Determine if the interface on which the frame is received is bound to a Virtual Circuit. If not, the packet is handled as per the steps enumerated earlier. If the incoming interface is bound to a Virtual Circuit, go to 2.

2. Check whether an FTN entry is associated with this interface. If yes, go to 3. If no, drop the packet and exit.

3. Determine whether the opcode associated with the FTN entry is either PUSH_AND_LOOKUP_FOR_VC or PUSH_FOR_VC. If it is the former, go to 4. Else go to 5.

4. Using the FTN entry bound to the interface, add an MPLS shim on top of the Ethernet frame received. Using the Virtual Circuit endpoint - the nexthop in the FTN entry - specified, carry out a lookup in the global FTN table for an FTN entry identifying an LSP from this node to the nexthop specified. If no entry is found, drop the packet. Else go to 6.

5. Using the FTN entry bound to the interface, add an MPLS shim on top of the Ethernet frame received, and pass on the shim+frame to the outgoing interface's Ethernet controller, so that this shim+frame can be encapsulated inside an Ethernet frame. Go to 7.

6. Using the newly found FTN entry, add to the existing shim that has been added on top of the received Ethernet frame, and then pass this shim+frame to the outgoing interface's Ethernet controller, so that this shim+frame can be encapsulated inside an Ethernet frame. Go to 7.

7. Forward the newly generated frame.

CHAPTER 4 FTN and VRF Label Interface APIs

This chapter contains the APIs for both FTN (FEC-To-NHLFE) and VRF (Virtual Routing Forwarding) label interface. It includes the following APIs:

Function	Description
ipi_mpls_clean_fib_for	Cleans up all of the ILM and FTN tables for entries that were populated by the specified protocol.
ipi_mpls_close_all_handles	Destroys the FTN and ILM tables in the MPLS Forwarder and closes the netlink socket.
ipi_mpls_debugging_handle	Turns debugging on and off
ipi_mpls_ftn4_entry_add	Adds the specified FTN entry to the FTN table.
ipi_mpls_ftn6_entry_add	Adds the specified IPv6 FTN entry to the FTN table.
ipi_mpls_ftn4_entry_del	Removes the specified entry from the FTN table.
ipi_mpls_ftn6_entry_del	Removes the specified IPv6 entry from the FTN table.
ipi_mpls_if_end	Disables MPLS Forwarding on the specified interface.
ipi_mpls_if_init	Enables MPLS Forwarding on the specified interface.
ipi_mpls_ilm4_entry_add	Adds the specified ILM entry to the ILM table.
ipi_mpls_ilm6_entry_add	Adds the specified IPv6 ILM entry to the ILM table.
ipi_mpls_ilm_entry_del	Removes the specified entry from the ILM table.
ipi_mpls_init_all_handles	Creates a netlink socket for the user-space process to communicate with the MPLS Forwarder.
ipi_mpls_local_pkt_handle	Enables or disables the labeling of locally generated TCP packets by the forwarder.
ipi_mpls_send_ttl	Sets the new TTL value for all packets switched through LSPs that use the current LSR for either the ingress or the egress.
ipi_mpls_clean_vrf_for	Cleans up all of the VRF tables for entries that were populated by the specified protocol.
ipi_mpls_if_update_vrf	Updates the VRF that the specified interface is bound to.
ipi_mpls_vrf_end	Unbinds the specified interface from the VRF table. I
ipi_mpls_vrf_init	Binds the specified interface to the specified VRF table.
ipi_mpls_vc_end	Unbinds the specified interface from the Virtual Circuit.

Function	Description
ipi_mpls_vc4_fib_add	Adds the specified IPv4 Virtual Circuit forwarding entry to the MPLS Forwarder.
ipi_mpls_vc6_fib_add	Adds the specified IPv6 Virtual Circuit forwarding entry to the MPLS Forwarder.
ipi_mpls_vc_fib_delete	Deletes the specified IPv4 or IPv6 Virtual Circuit forwarding entry from the MPLS Forwarder.
ipi_mpls_vc_init	Binds the specified Virtual Circuit to the specified interface.

Incoming Label Mapping/FTN APIs

The MPLS Forwarder package provides a static library (`libmpls_client.a`) that can be linked to a user-space process to populate the FTN and ILM tables in the MPLS Forwarder.

ipi_mpls_clean_fib_for

This function cleans up all of the ILM and FTN tables for entries that were populated by the specified protocol.

Syntax

```
int
ipi_mpls_clean_fib_for (u_char protocol)
```

Input Parameters

`protocol` Protocol that owns the entries:

`IPI_PROTO_NSM`

`IPI_PROTO_LDP`

`IPI_PROTO_BGP`

`IPI_PROTO_RSVP`

Output Parameters

No data is returned by this call.

Return Value

The status of this call is 0 if the call is successful; the status of this call is -1 if the FIB cannot be cleaned.

ipi_mpls_close_all_handles

This function destroys the FTN and ILM tables in the MPLS forwarder and closes the created netlink socket. The native IP packet handler processes all IP packets and drops MPLS packets. Subsequent executions of this call are ignored.

Syntax

```
int
ipi_mpls_close_all_handles (u_char protocol)
```

Input Parameters

`protocol`	Protocol that owns the entries:
	`IPI_PROTO_NSM`
	`IPI_PROTO_LDP`
	`IPI_PROTO_BGP`
	`IPI_PROTO_RSVP`

Output Parameters

No data is returned by this call.

Return Value

The status of this call is 0 when call is successful; it returns -1 when it cannot send the close signal to MPLS Forwarder.

ipi_mpls_debugging_handle

This function turns debugging on and off.

Syntax

```
int
ipi_mpls_debugging_handle (u_char protocol, u_char msg_type,
                           int enable)
```

Input Parameters

`protocol`	Protocol that owns the entries:
	`IPI_PROTO_NSM`
	`IPI_PROTO_LDP`
	`IPI_PROTO_BGP`
	`IPI_PROTO_RSVP`

`msg_type`	Error warning debug notice
`enable`	True or false

Return Value

The status of this call is 0 when call is successful; it returns -1 when it cannot send the close signal to the MPLS Forwarder.

ipi_mpls_ftn4_entry_add

This function adds the specified FTN entry to the FTN table. If this entry already exists in the FTN table, a warning is logged, and the request is ignored. The addition of an FTN entry requires the specification of the protocol adding it, as well as the operational code that the MPLS Forwarder should use when treating a packet pertaining to this FTN entry. This function can also be used to modify an existing FTN entry.

Syntax

```
int
ipi_mpls_ftn4_entry_add (int vrf,
```

```
                        u_char protocol,
                        struct pal_in4_addr *fec_addr,
                        u_char *prefixlen,
                        u_char *dscp_in,
                        u_int32_t *tunnel_label,
                        struct pal_in4_addr *tunnel_nhop,
                        struct if_ident *tunnel_if_info,
                        u_int32_t *vpn_label,
                        struct pal_in4_addr *vpn_nhop,
                        struct if_ident *vpn_if_info,
                        u_int32_t *tunnel_id,
                        u_int32_t *qos_resrc_id,
#if (!defined (HAVE_GMPLS) || defined (HAVE_PACKET))
#ifdef HAVE_DIFFSERV
                        struct ds_info_fwd *ds_info,
#endif /* HAVE_DIFFSERV */
#endif /* !HAVE_GMPLS || HAVE_PACKET */
                        char opcode,
                        u_int32_t nhlfe_ix,
                        u_int32_t ftn_ix,
                        u_char ftn_type,
                        struct mpls_owner_fwd *owner,
                        u_int32_t bypass_ftn_ix,
#ifdef HAVE_MPLS_TP
                        u_int32_t tunnel_ix,
                        u_int8_t *nh_mac,
#endif/* HAVE_MPLS_TP */
                        u_char lsp_type,
                        int active_head)
```

Input Parameters

`vrf`	Identifier for the VRF table that this FTN entry should be added to. A value of -1 specifies that the FTN entry should be added to the Global FTN table.

`protocol`	Protocol that owns the entries:
	`IPI_PROTO_NSM`
	`IPI_PROTO_LDP`
	`IPI_PROTO_BGP`
	`IPI_PROTO_RSVP`

`fec_addr`	IP address of FEC corresponding to this FTN entry.
`prefixlen`	Length of the prefix for this FEC.
`dscp_in`	DiffServ code point of the incoming IP packet.
`tunnel_label`	Outgoing label ID of the tunnel
`tunnel_nhop`	IP address of the next-hop to be used for this FEC
`tunnel_if_info`	IP address of the identifying object for the outgoing interface for this tunnel
`vpn_label`	Outgoing label ID of the VPN
`vpn_nhop`	IP address of the next-hop to be used for this FEC

vpn_if_info	IP address of the identifying object for the outgoing interface for the VPN
tunnel_id	Tunnel identifier of the LSP.
qos_resrc_id	Identifier of the QOS resource associated with the LSP.
ds_info	Diffserv related LSP information. This parameter is available only if the DiffServ compilation option is enabled.
opcode	Add FTN_LOOKUP to the existing operational code set.
nhlfe_ix	NHLFE index of the FTN entry.
ftn_ix	FTN index of the entry in the user space process.
ftn_type	MPLS FTN type
owner	Information specific to the signalling protocol module.
bypass_ftn_ix	Bypass FTN index
lsp_type	LSP type
tunnel_ix	Tunnel index
*nh_mac	Nexthop MAC address
active_head	Active head

Output Parameters

No data is returned by this call.

Return Value

This call returns 0 if it successfully adds the entry.

The status of this call is -1 if any of these parameters contains NULL or 0 data: `FEC_ADDR`, `FEC_PREFIX_LEN`, `NEXTHOP_ADDR` or `IFNAME`; or if the call to the ILM entry fails.

ipi_mpls_ftn6_entry_add

This function adds the specified IPv6 FTN entry to the FTN table. If this entry already exists in the FTN table, a warning is logged, and the request is ignored. The addition of an FTN entry requires the specification of the protocol adding it, as well as the operational code that the MPLS Forwarder should use when treating a packet pertaining to this FTN entry. This function can also be used to modify an existing IPv6 FTN entry.

Syntax

```
int
ipi_mpls_ftn6_entry_add (int vrf,
                         u_char protocol,
                         struct pal_in4_addr *fec_addr,
                         u_char *prefixlen,
                         u_char *dscp_in,
                         u_int32_t *tunnel_label,
                         struct pal_in4_addr *tunnel_nhop,
                         struct if_ident *tunnel_if_info,
                         u_int32_t *vpn_label,
                         struct pal_in4_addr *vpn_nhop,
                         struct if_ident *vpn_if_info,
                         u_int32_t *tunnel_id,
```

```
                        u_int32_t *qos_resrc_id,
#if (!defined (HAVE_GMPLS) || defined (HAVE_PACKET))
#ifdef HAVE_DIFFSERV
                        struct ds_info_fwd *ds_info,
#endif /* HAVE_DIFFSERV */
#endif /* !HAVE_GMPLS || HAVE_PACKET */
                        char opcode,
                        u_int32_t nhlfe_ix,
u_char lsp_type)
 struct mpls_owner_fwd *owner,
                        u_int32_t bypass_ftn_ix,
                        u_char lsp_type,
                        int active_head)
```

Input Parameters

`vrf`	Identifier for the VRF table that this IPv6 FTN entry should be added to. A value of -1 specifies that the FTN entry should be added to the Global FTN table.

`protocol`	Protocol that owns the entries
	`IPI_PROTO_NSM`
	`IPI_PROTO_LDP`
	`IPI_PROTO_BGP`
	`IPI_PROTO_RSVP`

`fec_addr`	IP address of FEC corresponding to this IPv6 FTN entry.
`prefixlen`	Length of the prefix for this FEC.
`dscp_in`	DiffServ code point of the incoming IP packet.
`tunnel_label`	Outgoing label ID of the tunnel
`tunnel_nhop`	IP address of the next-hop to be used for this FEC
`tunnel_if_info`	IP address of the identifying object for the outgoing interface for this tunnel
`vpn_label`	Outgoing label ID of the VPN
`vpn_nhop`	IP address of the next-hop to be used for this FEC
`vpn_if_info`	IP address of the identifying object for the outgoing interface for the VPN
`tunnel_id`	Tunnel identifier of the LSP.
`qos_resrc_id`	Identifier of the QOS resource associated with the LSP.
`ds_info`	Diffserv related LSP information. This parameter is available only if the DiffServ compilation option is enabled.
`opcode`	Add FTN_LOOKUP to the existing operational code set.
`nhlfe_ix`	NHLFE index of the IPv6 FTN entry.
`ftn_ix`	FTN index of the entry in the user space process.
`ftn_type`	MPLS IPv6 FTN type
`owner`	Information specific to the signalling protocol module.
`bypass_ftn_ix`	Bypass IPv6 FTN index
`lsp_type`	LSP type
`ttunnel_ix`	Tunnel index

*nh_mac	Nexthop MAC address
active_head	Active head

Output Parameters

No data is returned by this call.

Return Value

This call returns 0 if it successfully adds the entry.

The status of this call is -1 if any of these parameters contains NULL or 0 data: FEC_ADDR, FEC_PREFIX_LEN, NEXTHOP_ADDR or IFNAME; or if the call to the ILM entry fails.

ipi_mpls_ftn4_entry_del

This function removes the specified entry from the FTN table. If this entry is not present in the FTN table, a warning is logged, and the request is ignored. If the protocol specified does not match the one stored in the FTN entry, the delete operation will fail.

Syntax

```
int
ipi_mpls_ftn4_entry_del (int vrf,
                         u_char protocol,
                         u_int32_t *tunnel_id,
                         struct pal_in4_addr *fec_addr,
                         u_char *prefixlen,
                         u_char *dscp_in,
                         u_int32_t nhlfe_ix,
                         struct pal_in4_addr *tunnel_nhop,
#ifdef HAVE_MPLS_TP
                          u_int32_t tunnel_ix,
#endif/* HAVE_MPLS_TP */
                         u_int32_t ftn_ix)
```

Input Parameters

vrf	Identifier for the VRF table that this FTN entry should be added to. A value of -1 specifies that the FTN entry should be added to the Global FTN table.
protocol	Protocol that owns the entries IPI_PROTO_NSM IPI_PROTO_LDP IPI_PROTO_BGP IPI_PROTO_RSVP
tunnel_id	Tunnel identifier of the LSP.
fec_addr	IP address of FEC corresponding to this ILM entry.
prefixlen	Length of the prefix for this FEC.
dscp_in	Diffserv code point of the incoming IP packet.
nhlfe_ix	NHLFE index of the FTN entry.

`tunnel_nhop`	IP address of the next-hop to be used for this FEC
`tunnel_ix`	Tunnel index.
`ftn_ix`	FTN index of the entry in the user space process.

Output Parameters

No data is returned by this call.

Return Value

The status of this call is 0 if it deletes the entry successfully.

The status of this call is -1 if the `fec_addr` or the `prefix_len` are 0; it returns an error if the netlink socket fails.

ipi_mpls_ftn6_entry_del

This function removes the specified IPv6 entry from the FTN table. If this entry is not present in the FTN table, a warning is logged, and the request is ignored. If the protocol specified does not match the one stored in the IPv6 FTN entry, the delete operation will fail.

Syntax

```
int
ipi_mpls_ftn6_entry_del (int vrf,
                         u_char protocol,
                         u_int32_t *tunnel_id,
                         struct pal_in4_addr *fec_addr,
                         u_char *prefixlen,
                         u_char *dscp_in,
                         u_int32_t nhlfe_ix,
                         struct pal_in4_addr *tunnel_nhop,
                         u_int32_t ftn_ix)
```

Input Parameters

`vrf`	Identifier for the VRF table that this IPv6 FTN entry should be added to. A value of -1 specifies that the FTN entry should be added to the Global FTN table.

`protocol`	Protocol that owns the entries
	`IPI_PROTO_NSM`
	`IPI_PROTO_LDP`
	`IPI_PROTO_BGP`
	`IPI_PROTO_RSVP`

`tunnel_id`	Tunnel identifier of the LSP.
`fec_addr`	IP address of FEC corresponding to this ILM entry.
`prefixlen`	Length of the prefix for this FEC.
`dscp_in`	Diffserv code point of the incoming IP packet.
`nhlfe_ix`	NHLFE index of the IPv6 FTN entry.
`tunnel_nhop`	IP address of the next-hop to be used for this FEC
`ftn_ix`	FTN index of the entry in the user space process.

Output Parameters

No data is returned by this call.

Return Value

The status of this call is 0 if it deletes the entry successfully.

The status of this call is -1 if the `fec_addr` or the `prefix_len` are 0; it returns an error if the netlink socket fails.

ipi_mpls_if_end

This function disables MPLS Forwarding on the specified interface.

Note: No further MPLS processing will occur through this interface, and all labeled packets received by this interface will be dropped.

Syntax

```
int
ipi_mpls_if_end (struct if_ident *if_info)
```

Input Parameters

`if_info`	Identifying object for the interface.

Output Parameters

No data is returned by this call.

Return Value

The status of this call is 0 if the interface is removed.

The status of this call is -1 if the interface is not removed.

ipi_mpls_if_init

This function enables MPLS Forwarding on the specified interface. If an MPLS packet is received on a non-MPLS interface, the packet is dropped.

Syntax

```
int
ipi_mpls_if_init (struct if_ident *if_info,
                  unsigned short label_space)
```

Input Parameters

`if_info`	Identifying object for the interface.
`label_space`	Value of the interface.

Output Parameters

No data is returned by this call.

Return Value

The status of this call is 0 if it successfully initializes forwarding on the interface.

The status of this call is -1 if the forwarding initialization fails.

ipi_mpls_ilm4_entry_add

This function adds the specified ILM entry to the ILM table. If this entry already exists in the ILM table, a warning is logged, and the request is ignored. The addition of an ILM entry requires the specification of the protocol adding it and the operational code that the MPLS Forwarder should use when treating a packet pertaining to this ILM entry. This function can also be used to modify an existing ILM entry.

Syntax

```
int ipi_mpls_ilm4_entry_add (u_char protocol,
                             u_int32_t *label_id_in,
                             u_int32_t *label_id_out,
                             struct if_ident *if_in,
                             struct if_ident *if_out,
                             struct pal_in4_addr *fec_addr,
                             u_char *fec_prefixlen,
                             struct pal_in4_addr *nexthop_addr,
                             u_char is_egress,
#if (!defined (HAVE_GMPLS) || defined (HAVE_PACKET))
#ifdef HAVE_DIFFSERV
                             struct ds_info_fwd *ds_info,
#endif /* HAVE_DIFFSERV */
#endif /* !HAVE_GMPLS || HAVE_PACKET */
                             char opcode,
                             u_int32_t *tunnel_label,
                             u_int32_t *qos_resource_id,
                             u_int32_t nhlfe_ix,
#ifdef HAVE_MPLS_TP
                             u_int8_t *nh_mac,
#endif/* HAVE_MPLS_TP */
                             struct mpls_owner_fwd *owner,
                             u_int32_t vpn_id,
                             struct pal_in4_addr *vc_peer)
```

Input Parameters

`protocol`	Protocol that owns the entries
	`IPI_PROTO_NSM`
	`IPI_PROTO_LDP`
	`IPI_PROTO_BGP`
	`IPI_PROTO_RSVP`
`label_id_in`	Incoming label ID. Only the low-order 20 bits are used.
`label_id_out`	Outgoing label ID. Only the low-order 20 bits are used.
`if_in`	Identifying object for the incoming interface
`if_out`	Identifying object for the outgoing interface.
`fec_addr`	IP address of the FEC corresponding to this ILM entry.
`fec_prefixlen`	Length of the prefix for this FEC.
`nexthop_addr`	IP address of the next-hop to be used for this FEC.

is_egress	Flag to identify whether the LSR is an egress for this FEC.
ds_info	Diffserv related LSP information. This parameter is available only if the DiffServ compilation option is enabled.
opcode	Operational code to be applied to this ILM entry.
	POP
	SWAP
	POP_FOR_VPN
tunnel_label	Outgoing label ID of the tunnel
qos_resource_id	
	QoS resource identifier
nhlfe_ix	NHLFE index of the ILM entry.
nh_mac	Nexthop MAC address
owner	Information specific to the signalling protocol module.
vpn_id	VPN ID
vc_peer	IP address of the Virtual Circuit peer

Output Parameters

No data is returned by this call.

Return Value

This call returns 0 if it successfully adds the entry.

The status of this call is -1 if any of the parameters contains NULL or 0 data, or if the call to the ILM entry fails.

ipi_mpls_ilm6_entry_add

This function adds the specified IPv6 ILM entry to the ILM table. If this entry already exists in the ILM table, a warning is logged, and the request is ignored. The addition of an IPv6 ILM entry requires the specification of the protocol adding it and the operational code that the MPLS Forwarder should use when treating a packet pertaining to this IPv6 ILM entry. This function can also be used to modify an existing IPv6 ILM entry.

Syntax

```
int ipi_mpls_ilm6_entry_add (u_char protocol,
                             u_int32_t *label_id_in,
                             u_int32_t *label_id_out,
                             struct if_ident *if_in,
                             struct if_ident *if_out,
                             struct pal_in6_addr *fec_addr,
                             u_char *fec_prefixlen,
                             struct mpls_nh_fwd *nexthop_addr,
                             u_char is_egress,
#if (!defined (HAVE_GMPLS) || defined (HAVE_PACKET))
#ifdef HAVE_DIFFSERV
                             struct ds_info_fwd *ds_info,
#endif /* HAVE_DIFFSERV */
#endif /* !HAVE_GMPLS || HAVE_PACKET */
```

```
                        char opcode,
                        u_int32_t *tunnel_label,
                        u_int32_t *qos_resource_id,
                        u_int32_t nhlfe_ix,
                        struct mpls_owner_fwd *owner,
                        u_int32_t vpn_id,
                         struct pal_in6_addr *vc_peer)
```

Input Parameters

protocol	Protocol that owns the entries
	IPI_PROTO_NSM
	IPI_PROTO_LDP
	IPI_PROTO_BGP
	IPI_PROTO_RSVP
label_id_in	Incoming label ID. Only the low-order 20 bits are used.
label_id_out	Outgoing label ID. Only the low-order 20 bits are used.
if_in	Identifying object for the incoming interface
if_out	Identifying object for the outgoing interface.
fec_addr	IP address of FEC corresponding to this IPv6 ILM entry.
fec_prefixlen	Length of the prefix for this FEC.
nexthop_addr	IP address of the next-hop to be used for this FEC.
is_egress	Flag to identify whether the LSR is an egress for this FEC.
ds_info	Diffserv related LSP information. This parameter is available only if the DiffServ compilation option is enabled.
opcode	Operational code to be applied to this IPv6 ILM entry.
	POP
	SWAP
	POP_FOR_VPN
tunnel_label	Outgoing label ID of the tunnel
*qos_resource_id	
	QoS resource identifier.
nhlfe_ix	NHLFE index of the ILM entry.
owner	Information specific to the signalling protocol module.
vpn_id	VPN ID
vc_peer	IP address of the Virtual Circuit peer

Output Parameters

No data is returned by this call.

Return Value

This call returns 0 if it successfully adds the entry.

The status of this call is -1 if any of the parameters contains NULL or 0 data, or if the call to the IPv6 ILM entry fails.

ipi_mpls_ilm_entry_del

This function removes the specified entry from the ILM table. If this entry is not present in the ILM table, a warning is logged, and the request is ignored. If the protocol specified does not match the one stored in the ILM entry, the delete operation fails.

Syntax

```
int
ipi_mpls_ilm_entry_del (u_char protocol,
                        u_int32_t *label_id_in,
                        struct if_ident *if_info)
```

Input Parameters

`protocol`	Protocol that owns the entries
	`IPI_PROTO_NSM`
	`IPI_PROTO_LDP`
	`IPI_PROTO_BGP`
	`IPI_PROTO_RSVP`

`label_id_in`	Incoming label ID. Only the low-order 20 bits are used.
`if_info`	Identifying object for incoming interface

Output Parameters

No data is returned by this call.

Return Value

The status of this call is 0 if it successfully removes the entry.

The status of this call is -1 if the incoming label is 0, if the IF_INDEX is a null pointer, or if the call to the ILM entry fails.

ipi_mpls_init_all_handles

This function creates a netlink socket for the user-space process to communicate with the MPLS Forwarder, and creates the FTN and ILM tables. The MPLS forwarder intercepts all packets of types IP and MPLS, and handles each appropriately. Subsequent executions of this function are ignored.

Syntax

```
int
ipi_mpls_init_all_handles (struct lib_globals *zg, u_char protocol)
```

Input Parameters

`*zg`	Pointer to the LIB globals.

`protocol`	Protocol that owns the entries
	`IPI_PROTO_NSM`
	`IPI_PROTO_LDP`
	`IPI_PROTO_BGP`
	`IPI_PROTO_RSVP`

Output Parameters

The status of this call is the socket.

Return Value

The status of this call is -1 when it cannot send the initial request to the Forwarder.

ipi_mpls_local_pkt_handle

This function enables or disables the labeling of locally generated TCP packets by the forwarder.

Syntax

```
int
ipi_mpls_local_pkt_handle (u_char protocol, int enable)
```

Input Parameters

`protocol`		Protocol that owns the entries
	`IPI_PROTO_NSM`	
	`IPI_PROTO_LDP`	
	`IPI_PROTO_BGP`	
	`IPI_PROTO_RSVP`	
`enable`		Flag to enable or disable forwarding of locally generated packets over LSPs.
	`TRUE`	Enable
	`FALSE`	Disable

Output Parameters

No data is returned by this call.

Return Value

The status of this call is 0 if the call is successful.

The status of this call is -1 if the named interface data could not be disabled.

ipi_mpls_send_ttl

This function sets the new TTL value for all packets switched through LSPs that use the current LSR for either the ingress or the egress. This function is executed by the NSM. See the *Network Services Module Command Reference* for more about the MPLS configuration utility. A value of `-1` for the new TTL causes the forwarder to use the default mechanism (copying the TTL from the IP packet to the labeled packet or vice versa).

Syntax

```
int
ipi_mpls_send_ttl (u_char protocol, u_char type, int ingress, int new_ttl)
```

Input Parameters

`protocol`		Protocol that owns the entries
	`IPI_PROTO_NSM`	

	IPI_PROTO_LDP
	IPI_PROTO_BGP
	IPI_PROTO_RSVP
type	TTL configuration information passed between NSM and MPLS forwarder:
	MPLS_TTL_VALUE_SET 1
	MPLS_TTL_PROPAGATE_SET 2
	MPLS_TTL_PROPAGATE_UNSET 3
	MPLS_TTL_MODEL_PIPE_SET 4
	MPLS_TTL_MODEL_PIPE_UNSET 5
ingress	Is ingress
new_ttl	The value of the new TTL.

Output Parameters

No data is returned by this call.

Return Value

The status of this call is 0 if the call is successful.

The status of this call is -1 if the named interface data could not change the TTL setting.

Virtual Routing Forwarding Label Interface APIs

The MPLS Forwarder package provides a static library `libmpls_client.a` that can be linked to a user-space process to populate the VRF tables in the MPLS Forwarder.

ipi_mpls_clean_vrf_for

This function cleans up all of the VRF tables for entries that were populated by the specified protocol.

Syntax

```
int
ipi_mpls_clean_vrf_for (u_char protocol)
```

Input Parameters

protocol	Protocol that owns the entries
	IPI_PROTO_NSM
	IPI_PROTO_BGP

Output Parameters

No data is returned by this call.

Return Value

The status of this call is 0 if the call is successful.

The status of this call is -1 if the VRF tables cannot be cleaned.

ipi_mpls_if_update_vrf

This function updates the VRF that the specified interface is bound to. Passing '-1' as the VRF ID unbinds the interface from the VRF that it was associated to. Once the interface is unbound from a VRF, all incoming IP packets on this interface are forwarded using the global FTN table.

Syntax

```
int
ipi_mpls_if_update_vrf (struct if_ident *if_info,
                        int vrf_ident)
```

Input Parameters

`if_info`	Interface data.
`vrf_ident`	Identifier for this VRF table.

Output Parameters

This call returns no data.

Return Value

The status of this call is -1 if the update could not be made. It is 0 otherwise.

ipi_mpls_vrf_end

This function unbinds the specified interface from the VRF table. If no more interfaces are bound to the VRF table in question, the VRF table is removed from the MPLS Forwarder.

Syntax

```
int
ipi_mpls_vrf_end (int vrf_ident)
```

Input Parameters

`vrf_ident`	Identifier for the virtual route forwarding table.

Output Parameters

No data is returned by this call.

Return Value

The status of this call is 0 if the VRF table is deleted.

The status of this call is -1 if the interface is not deleted.

ipi_mpls_vrf_init

This function binds the specified interface to the specified VRF table. If this VRF table does not exist, a new one is created. An interface might be VRF enabled, even if it is not MPLS enabled. For each packet subsequently received by this interface, the Forwarder first tries to find a match for the specified FEC in the VRF table, and then finds a match in the global FTN table, if required.

Syntax

```
int
ipi_mpls_vrf_init (int vrf_ident)
```

Input Parameters

`vrf_ident`	Identifier for the Virtual Route Forwarding table.

Output Parameters

No data is returned by this call.

Return Value

The status of this call is 0 if the VRF table is created.

The status of this call is -1 if the interface is not created.

Virtual Circuit Interface APIs

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.

ipi_mpls_vc_end

This function unbinds the specified interface from the Virtual Circuit.

Syntax

```
int
ipi_mpls_vc_end (u_int32_t vc_id, struct if_ident *if_info,
                 u_int32_t vlan_id)
```

Input Parameters

`vc_id`	Virtual Circuit ID
`*if_info`	Identifying object for interface to which the Virtual Circuit was bound.
`vlan_id`	VLAN ID

Output Parameters

No data is returned by this call.

Return Value

The status of this call is 0 if the Virtual Circuit binding was removed from the interface.

The status of this call is -1 if the Virtual Circuit binding was not removed from the interface.

ipi_mpls_vc4_fib_add

This function adds the specified IPv4 Virtual Circuit forwarding entry to the MPLS Forwarder.

Syntax

```
int
ipi_mpls_vc4_fib_add (u_int32_t vc_id,
                      u_int32_t vc_style,
                      u_int32_t *vpls_id,
                      u_int32_t *in_label,
                      u_int32_t *out_label,
                      u_int32_t *ac_if_ix,
                      u_int32_t *nw_if_ix,
                      struct if_ident *if_in,
                      struct if_ident *if_out,
                      struct if_ident *if_tnl_in,
                      u_char opcode,
                      struct pal_in4_addr *peer_addr,
                      struct pal_in4_addr *peer_nhop_addr,
                      struct pal_in4_addr *fec_addr,
                      u_char *fec_prefixlen,
                      u_int32_t *tunnel_label,
                      struct pal_in4_addr *tunnel_nhop,
                      u_int32_t *tunnel_oix,
                      u_int32_t *tunnel_nhlfe_ix,
                      u_int32_t *tunnel_ftnix,
#ifdef HAVE_MPLS_TP
                      u_int32_t vlan_id,
                      u_int8_t *nh_mac,
#endif/* HAVE_MPLS_TP */
                      u_int8_t is_ms_pw)
```

Input Parameters

vc_id	Identifier for the Virtual Circuit being configured.
vc_style	Virtual Circuit style
	MARTINI
	VPLS MESH
	VPLS SPOKE
vpls_id	VPLS identifier
in_label	Incoming label ID
out_label	Outgoing label ID
ac_if_ix	Virtual Circuit bounded access interface index
nw_if_ix	Virtual Circuit network side interface index
if_in	Identifying object for the incoming interface
if_out	Identifying object for the outgoing interface
if_tnl_in	Identifying object for the tunnel interface

`opcode`	Virtual Circuit forwarding entry operational code PUSH_FOR_VC PUSH_AND_LOOKUP_FOR_VC
`peer_addr`	Peer address for Virtual Circuit
`peer_nhop_addr`	Next-hop address to reach the Virtual Circuit peer
`fec_addr`	Prefix address for referring the ILM entry in the MPLS forwarder
`fec_prefixlen`	FEC address prefix length
`tunnel_label`	Virtual Circuit tunnel label
`tunnel_nhop`	Next-hop of the Virtual Circuit tunnel
`tunnel_oix`	Outgoing interface index of the Virtual Circuit tunnel FTN
`tunnel_nhlfe_ix`	NHLFE index of the Virtual Circuit tunnel FTN
`*tunnel_ftnix`	Pass tunnel label
`vlan_id`	VLAN ID
`*nh_mac,`	Nexthop MAC address
`is_ms_pw`	

Output Parameters

No data is returned by this call.

Return Value

The status of this call is 0 if the Virtual Circuit binding was removed from the interface.

The status of this call is -1 if the Virtual Circuit binding was not removed from the interface.

ipi_mpls_vc6_fib_add

This function adds the specified IPv6 Virtual Circuit forwarding entry to the MPLS Forwarder.

Syntax

```
int
ipi_mpls_vc6_fib_add (u_int32_t vc_id,
                      u_int32_t vc_style,
                      u_int32_t *vpls_id,
                      u_int32_t *in_label,
                      u_int32_t *out_label,
                      u_int32_t *ac_if_ix,
                      u_int32_t *nw_if_ix,
                      struct if_ident *if_in,
                      struct if_ident *if_out,
                      u_char opcode,
                      struct pal_in4_addr *peer_addr,
                      struct pal_in4_addr *peer_nhop_addr,
                      struct pal_in4_addr *fec_addr,
                      u_char *fec_prefixlen,
                      u_int32_t *tunnel_label,
                      struct pal_in4_addr *tunnel_nhop,
```

```
                u_int32_t *tunnel_oix,
                u_int32_t *tunnel_nhlfe_ix,
                u_int32_t *tunnel_ftnix)
```

Input Parameters

vc_id	Identifier for the Virtual Circuit being configured.
vc_style	Virtual Circuit style
	MARTINI
	VPLS MESH
	VPLS SPOKE
vpls_id	VPLS identifier
in_label	Incoming label ID
out_label	Outgoing label ID
ac_if_ix	Virtual Circuit bounded access interface index
nw_if_ix	Virtual Circuit network side interface index
if_in	Identifying object for the incoming interface
if_out	Identifying object for the outgoing interface
opcode	Virtual Circuit forwarding entry operational code
	PUSH_FOR_VC
	PUSH_AND_LOOKUP_FOR_VC
peer_addr	Peer address for the Virtual Circuit
peer_nhop_addr	Next-hop address to reach the Virtual Circuit peer
fec_addr	Prefix address for referring the ILM entry in the MPLS forwarder
fec_prefixlen	FEC address prefix length
tunnel_label	Virtual Circuit tunnel label
tunnel_nhop	Next-hop of the Virtual Circuit tunnel
tunnel_oix	Outgoing interface index of the Virtual Circuit tunnel FTN
tunnel_nhlfe_ix	
	NHLFE index of the Virtual Circuit tunnel FTN
tunnel_ftnix	Pass tunnel label

Output Parameters

No data is returned by this call.

Return Value

The status of this call is 0 if the Virtual Circuit binding was removed from the interface.

The status of this call is -1 if the Virtual Circuit binding was not removed from the interface.

ipi_mpls_vc_fib_delete

This function deletes the specified IPv4 or IPv6 Virtual Circuit forwarding entry from the MPLS Forwarder.

Syntax

```
int
ipi_mpls_vc_fib_delete (u_int32_t vc_id,
                        u_int32_t vc_style,
                        u_int32_t *vpls_id,
                        u_int32_t *in_label,
                        u_int32_t *out_label,
                        u_int32_t *ac_if_ix,
                        u_int32_t *nw_if_ix,
                        struct if_ident *if_in,
                        struct if_ident *if_out,
                        struct if_ident *if_tnl_in,
                        struct pal_in4_addr *peer_addr,
#ifdef HAVE_MPLS_TP
                        u_int32_t vlan_id,
#endif/* HAVE_MPLS_TP */
                        u_int8_t is_ms_pw)
```

Input Parameters

vc_id	Identifier for the Virtual Circuit being configured.
vc_style	Virtual Circuit style
	MARTINI
	VPLS MESH
	VPLS SPOKE
vpls_id	VPLS identifier
in_label	Incoming label ID
out_label	Outgoing label ID
ac_if_ix	Virtual Circuit bounded access interface index
nw_if_ix	Virtual Circuit network side interface index
if_in	Identifying object for the incoming interface
if_out	Identifying object for the outgoing interface
if_tnl_in	Identifying object for the tunnel interface
peer_addr	Peer address for Virtual Circuit
vlan_id	VLAN ID
is_ms_pw	

Output Parameters

No data is returned by this call.

Return Value

The status of this call is 0 if it deletes the entry successfully.

The status of this call is an error if the netlink socket fails.

ipi_mpls_vc_init

This function binds the specified Virtual Circuit to the specified interface. An interface can 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 of the packets received on the same interface are tunneled over the corresponding Virtual Circuit. If there is no Virtual Circuit FTN entry associated with this interface, the MPLS Forwarder drops all packets on this interface.

Syntax

```
int
ipi_mpls_vc_init (u_int32_t vc_id, struct if_ident *if_info,
                  u_int32_t vlan_id)
```

Input Parameters

`vc_id`	Identifier for the Virtual Circuit that is being bound to this interface
`if_info`	Identifying object for interface
`vlan_id`	VLAN identifier

Output Parameters

No data is returned by this call.

Return Value

The status of this call is 0 if it successfully binds the interface to the specified Virtual Circuit ID.

The status of this call is -1 if the binding of the interface to the specified Virtual Circuit ID fails.

Index