Next Hop LFB and Functional API Implementation Agreement

April 6, 2005
Revision 1.0

Editor:

Reda Haddad, Ericsson { Reda.Haddad@Ericsson.com }

The words "MUST", "MUST NOT", "REQUIRED", "SHALL", "SHALL NOT", "SHOULD", "SHOULD NOT", "RECOMMENDED", “MAY", and "OPTIONAL" in the remainder of this document are to be interpreted as described in the NPF Software API Conventions Implementation Agreement revision 1.0.

For additional information contact:
The Network Processing Forum, 39355 California Street,
Suite 307, Fremont, CA 94538
+1 510 608-5990 phone ✦ info@npforum.org

Table of Contents

Table of Figures

1 Revision History

V00	01/20/2004	Baseline text for Next Hop LFB/API
V01	01/27.2004	Changed the sources to “Reda Haddad” and added a comment to handle exceptions (default handling) Update to latest template doc (2003.407.05) Clearly stated that TTL check is optional Added an exception for TTL = 0 Added TTL as metadata Added figure to show RPF usage

		Added Appendix to show missing functionality in IPv4 SAPI
V02	09/28/2004	Added comments resolutions
V03	09/28/2004	Updated table of contents
V04	10/25/2004	Incorporate more comments resolutions
V05	10/28/2004	Change the LFB query to return the conservative estimate of the maximum number of tables and entries instead of the min (as requested by the original ballot comment submitter)
V06	10/29/2004	Added a caution on he usage of MTU in the Next Hop LFB.
V07	04/03/2005	Incorporate comments resolutions
V08	04/05/2005	Add ingress vif id to exc metadata, and remove the MTU scalability paragraph.
V09	04/05/2005	Make the ingress vif id more explicit in the exc metadata
V10	04/05/2005	Fit to LFB template and add ack section Change the dscp and weight policies sections. Reword section 1.4 Remove metadata types from section 2
V11	4/6/2005	Replaced title page, repositioned Revision History, added list of participating companies.

2 Introduction

Depending on the Forwarding plane NP architecture, a Forwarding Information Base (FIB) may be modeled in several ways. One mode, the unified table model, uses a single table for structuring and managing IPv4 unicast forwarding information. A second mode, the discrete table model, uses separate Prefix and Next Hop tables for structuring and managing IPv4 unicast forwarding information [IPV4SAPI].

This LFB relates to the discrete table model, specifically for the management of the Next Hop tables. It is responsible for selecting a Next Hop from a possible set of Next Hops associated with a prefix according to a preset policy (for example load balancing, MPLS LLSPs, …). It is also responsible for checking if the egress interface MTU allows the current packet to be forwarded without fragmentation (might trigger an ICMP packet if the DF bit is set and the MTU is too small). It also optionally maintains the TTL value associated with the Next Hop entry, in the case where TTL check is enabled.

The FAPI Logical Function Block APIs are used to configure LFB resources and associate resources between LFBs. This contribution describes the Next Hop LFB and its associated APIs.

2.1 Acronyms / Definitions

The following are acronyms used in this document.

API	**Application Programming Interface**
DF	**Don’t Fragment**
FAPI	**Functional API**
FIB	**Forwarding Information Base**
ICMP	**Internet Control Message Protocol**
LFB	**Logical Function Block**
LLSP	**Label-Only-Inferred-PSC Label Switched Path**
MPLS	**Multi-Protocol Label Switching**
MTU	**Maximum Transfer Unit**
RPF	**Reverse Path Forwarding**
TTL	**Time To Live**

2.2 Assumptions

There will be a separate Prefix LFB defined [IPV4FAPI] which will generate some of the metadata needed by the Next Hop LFB defined in this document.

2.3 Scope

This contribution concentrates on the details of the Next Hop LFB and APIs. It is intended to be used along with the Prefix LFB as part of the discrete model. A separate contribution will handle any details of the unified model LFB.

2.4 External Requirements and Dependencies

This API depends on the Software API Conventions [SWAPICON] and on the FAPI Topology Discovery API [FAPITOPO]. This API was designed in view of the requirements set by the ForCES WG [FORCESREQ] in the IETF.

3 LFB Detailed Description

[RFC1812] describes the steps involved in forwarding an IPv4 packet. Briefly, the router receives the IP packet from the link layer, validates the IP header, processes any IP options tagged with the IPv4 header, and examines the destination IP address to determine how it should continue to process the IP datagram. The last step may result in the packet being queued for local delivery, for forwarding, or for forwarding with a copy of the packet being queued for local delivery. This last step can be broken into two functionalities, lookup and forwarding. The lookup step is basically performing the prefix search usually on the destination IP address in the packet and is covered in the Prefix LFB/FAPI document. The forwarding step selects the next hop where the packet needs to be forwarded from a set of one or more next hop structures associated with the matched prefix,. This LFB class, shown in Figure 1, manages the Next Hop tables along with the Next Hop entities such as Next Hop Selector and Next Hop structures defined in this document.

The content of the selected Next Hop structure depends on the type of Next Hop. For example, an IPv4 next hop structure contains an IPv4 address and a reference to an egress interface used to forward the packet. A Next Hop structure also determines whether a packet needs to be forwarded to a locally attached host or a directly connected next hop router, or identifies that the packet has reached its destination.
[RFC2991] and [RFC2992] identify the need for load balancing or equal cost multipath along with ways to implement them. MPLS LLSPs [RFC3270] require the need for associating different DSCP values with different Next Hops. All of the above are supported by a Next Hop Selector grouping a set of Next Hops with an associated policy to select one Next Hop from the set. The Selector structure allows for fast BGP reroute as shown in this document. This document defines the usage and the management of the Next Hop Selector structures.

In addition to managing next hops, this LFB optionally performs Reverse Path Forwarding (RPF) check. RPF is used for unicast traffic as a mechanism to block source address spoofing [RFC2827]. When receiving a packet on the RPF ingress port (the port that performs the RPF functionality), this LFB checks whether at least one of the outgoing interfaces specified in the matched next hop entity, identified from the next hop entity id passed from the prefix LFB (the lookup being done on the source IP address), matches the incoming interface (RPF checking does not work well when asymmetric routing is allowed – it is usually performed at the edge of the network). If the packet's RPF check passes, the packet is then forwarded on the RPF egress LFB port which may loop back into the Next Hop LFB (the existence of intermediate LFBs before looping back is also possible) for normal next hop lookup based on the packet's destination address (Figure 3 illustrates the RPF description).
If the packet's RPF check fails, an exception is triggered and the packet is forwarded on the configured egress LFB port. Since RPF is optional, the ingress RPF port, if available as detected by topology scan, can be left unconnected if this functionality is to be skipped.

[RFC1812] mentions that a router cannot check and decrement the TTL before checking whether the packet should be delivered to the router itself. In general, the earliest stage of detecting whether a packet needs to be forwarded or terminated is the Next Hop LFB. This LFB optionally performs TTL decrement and TTL check, and issues an exception if the TTL reaches 0.

Finally, this LFB performs egress interface MTU checks. If the packet's length is larger than the associated next hop MTU, this LFB further checks the packet's DF bit. If the bit is set, a 'packet too big' exception is generated. Otherwise, if the bit is not set, a 'fragmentation required' exception is generated.

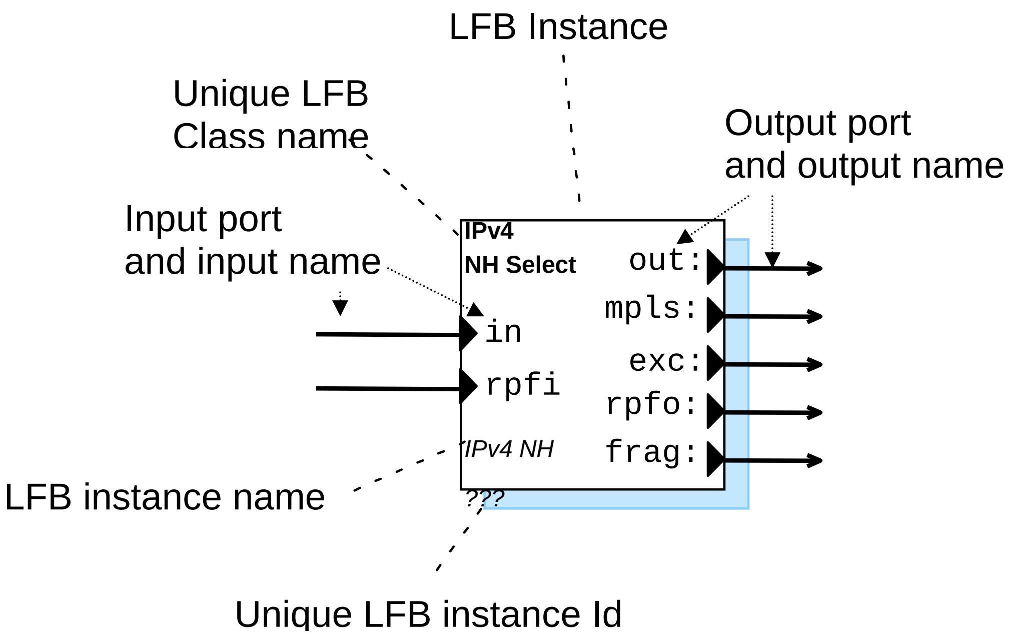

Figure 1 Generic LFB Diagram

The block type code for the Next hop LFB is:

```
#define NPF_IPv4_NEXTHOP_LFB        11
```

3.1 LFB Input Ports

This LFB has two input ports labeled 'in' and 'rpfi':

- The 'in' input port, numerically labeled port 1, is the normal packet input and expects any frame type (e.g. IPv4, IPv6, MPLS, etc.). Metadata, received along with packets on this port, will be used to determine the next hop selector/entry, and the embedded TTL and MTU values needed for the TTL and MTU checks, respectively. Packets with matched non-MPLS next hops will be forwarded to the output port 'out'. Packets with matched MPLS next hops will be forwarded to the output port 'mpls', Packets with length larger than the embedded next hop entry's MTU will be forwarded to the output port 'frag'. Any exceptions (drop, icmp, etc.) will be forwarded to the output port 'exc'.

- The 'rpfi' input port, numerically labeled port 2, is the RPF packet input and expects any frame type (e.g., IPv4, IPv6, MPLS, etc.). Metadata, received along with packets on this port, will be used to determine whether this packet fails the RPF check. If a packet fails the RPF check, an exception will be generated on output port ‘exc’. Otherwise, the packet is forwarded to output port ‘rpfo’.

3.1.1 Metadata Required

The ‘in’ port requires the following metadata:

- Next Hop Table Id (as defined in section 4.1.21)
- Next Hop Entity Id (as defined in section 4.1.13)

The ‘rpfi’ port requires the following metadata:

- ingress vif id. Note that it is assumed that ingress and egress VIF IDs are identical for the same network interface.
- Next Hop Table Id (as defined in section 4.1.21)
- Next Hop EntityId (as defined in section 4.1.13)

Note: the rpfi input next hop ids are derived based on the source ip address.

3.1.2 Optional Metadata

The ‘in’ port optionally receives:

- The derived_DSCP: (an upstream derived DSCP value when the DSCP is not present in the packet’s header e.g. for MPLS encapsulated packets).

3.2 LFB Output Ports

This LFB has five output ports labeled ‘out’, ‘mpls’, ‘exc’, ‘rpfo’ and ‘frag’. The metadata produced on each port is described in next subsection.

- The ‘out’ output port, numerically labeled port 1, is the normal packet output. The frame type associated with this port is the same as the frame type received on the ‘in’ input port.

- The ‘mpls’ output port, numerically labeled port 2, is the MPLS packet output. If a next hop identifies that the packet is to be forwarded as an MPLS packet, this port is used to insert the packet in the MPLS LFBs forwarding path versus the normal packet LFBs forwarding path. The frame type associated with this port is the same as the frame type received on the ‘in’ input port (can be IPv4, IPv6, MPLS).

- The ‘exc’ output port, numerically labeled port 3, is the exception packet output. Packets are forwarded on this port when an exception occurs, for example when the next hop id is not valid or when a packet needs to be dropped or when an icmp packet needs to be generated or when a packet is to be forwarded to CP for further processing or local termination. The frame type associated with this port is the same as the frame type received on the ‘in’ input port (or on the ‘rpfi’ input port in the case when the packet is received originally on the ‘rpfi’ input port).

- The ‘rpfo’ output port, numerically labeled port 4, is the RPF packet output. Packets received on the ‘rpfi’ input port will be forwarded on this output port. The frame type associated with this port is the same as the frame type received on the ‘in’ input port.

- The ‘frag’ output port, numerically labeled port 5, is the “fragmentation needed” packet output. A packet is forwarded on this output port when the packet, originally received on the ‘in’ input port, fails the MTU check (assuming the check is enabled) and needs to be fragmented before being forwarded on the normal LFB processing path. The frame type associated with this port is the same as the frame type received on the ‘in’ input port (note that some frames are not allowed to be fragmented, for which case an exception will be triggered instead).

3.2.1 Metadata Produced

The ‘out’ port emits the following metadata:

- egress vif id
- IPv4 Address or IPv6 Address (depending whether the next hop is IPv4 or IPv6). Note depending on whether the next hop is BASIC or DIRECT (see section 4.1.6) this address is derived either from the next hop entry or the packet’s destination address.
- The TTL value

The ‘mpls’ port emits the following metadata:

- egress vif id
- NHLFE Entry id (note: this type should be defined in the NHLFE LFB)
- The TTL value

The ‘exc’ port emits the following metadata:

- egress vif id IF available
- ingress vif id (passed through with the packet)
- Next Hop Exception (exception code, as defined in section 4.1.25)

The ‘rpfo’ port emits no metadata.

The ‘frag’ port emits the following metadata:

- egress vif id
- IPv4 Address (IPv4 next hop address, IPv6 does not allow fragmentation). Note depending on whether the next hop is BASIC or DIRECT (see section 4.1.6) this address is derived either from the next hop entry or the packet’s destination address.
- The TTL value
- The egress MTU value

3.3 Relationship with other LFBs

The IPv4 Prefix and Next Hop LFBs work in conjunction with other LFBs such as Egress VIF and Packet Handler LFBs as shown in Figure 2, in order to properly forward the packet. The figure does not force or dictate an implementation but rather gives an example of where the Next Hop LFB may be situated.
The IPv4 Prefix LFB performs the prefix lookup on the packet’s destination address and passes the Next Hop Id and the next hop table id metadatas to the Next Hop LFB. The Packet Handler

LFB handles the delivery of local or exception packets to the local/remote protocol stacks. The Egress VIF LFB carries information about the Egress Virtual Interface and its characteristics.

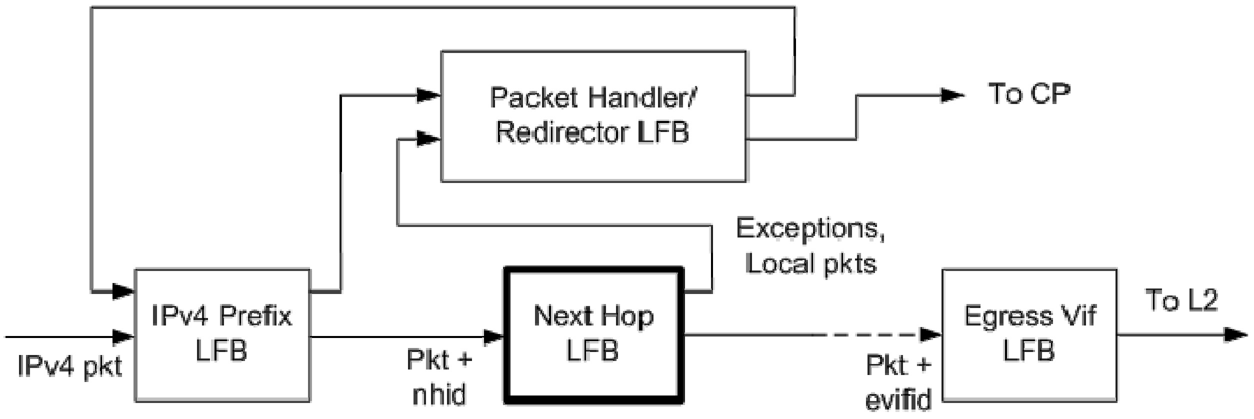

Figure 2 Relationship with other LFBs

Figure 2 shows the usage of the Next Hop LFB with the RPF input disconnected or not used. Figure 3 shows an example of how the RPF input can be connected to make use of the RPF functionality offered by this LFB.

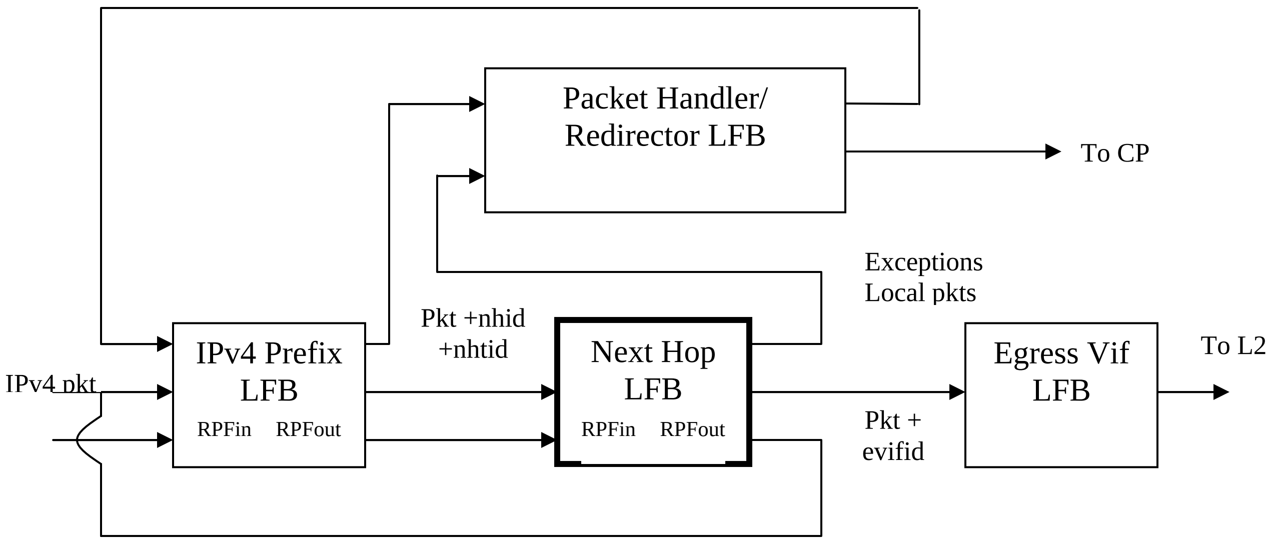

Figure 3 NH LFB with RPF input connected

Figure 4 shows an example of how the MPLS output port can be connected to an NHLFE LFB for the case of an MPLS FTN.

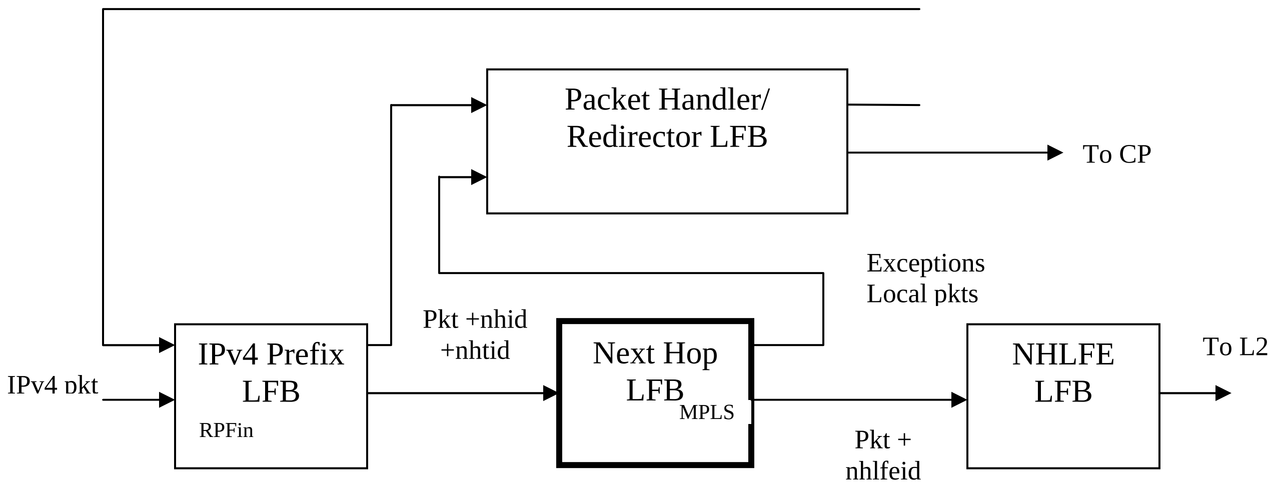

Figure 4 NH LFB with RPF input connected

4 Data Types

This section describes the Next Hop FAPI data structures definitions. A Next Hop Selector Structure, which possibly holds a number of Next Hop handles, is presented to handle load balancing, and MPLS cases. Both a Next Hop and a Next Hop Selector are located by a prefix entry in a Prefix table (within the prefix LFB). Figure 5 depicts a one table approach where both Next Hop Selector and Next Hop structures are embedded in the same table.

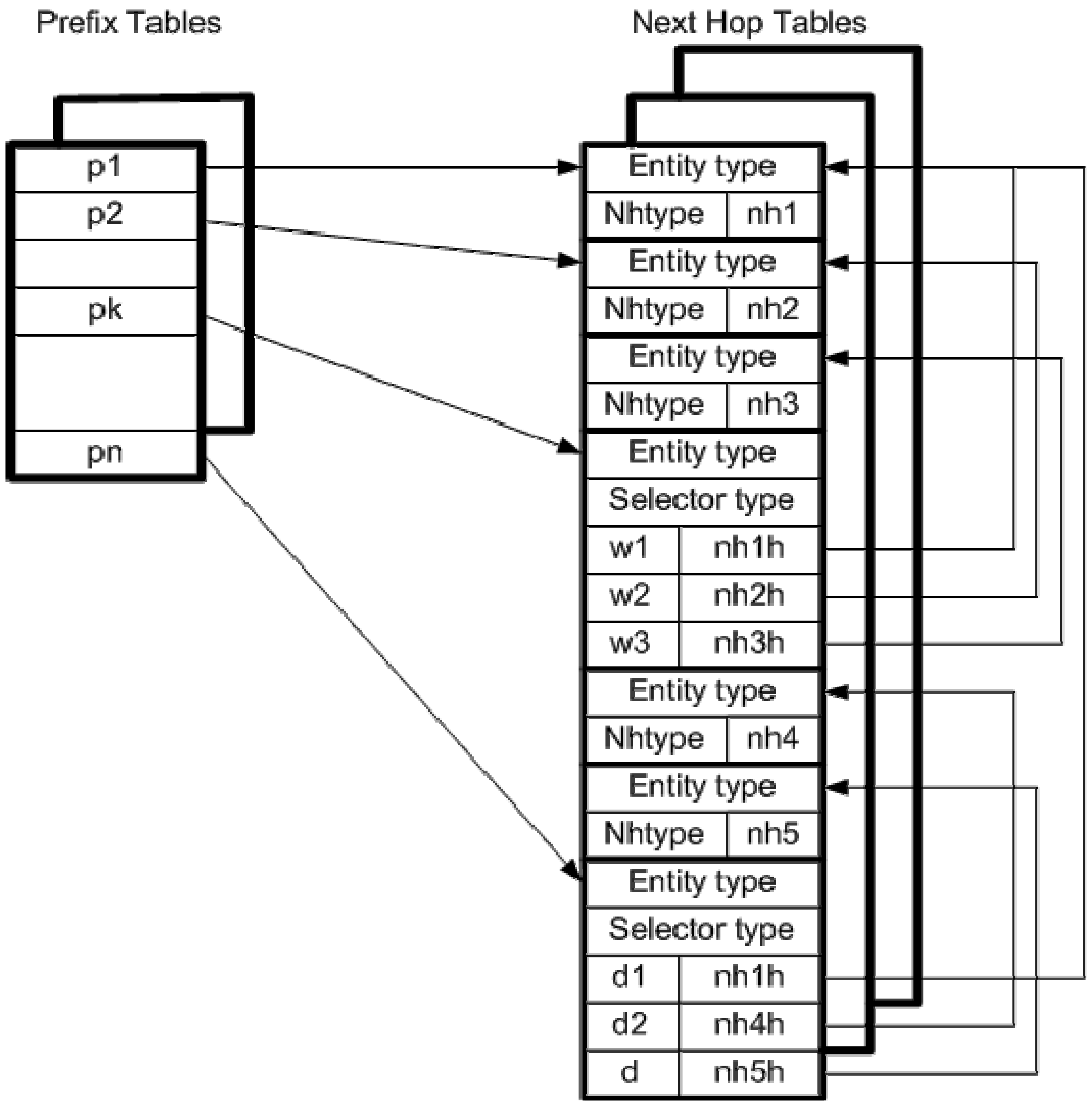

Figure 5 One table approach

Figure 6 depicts a two table approach where Next Hop Selectors are in one table and the Next Hop Structures are in another.

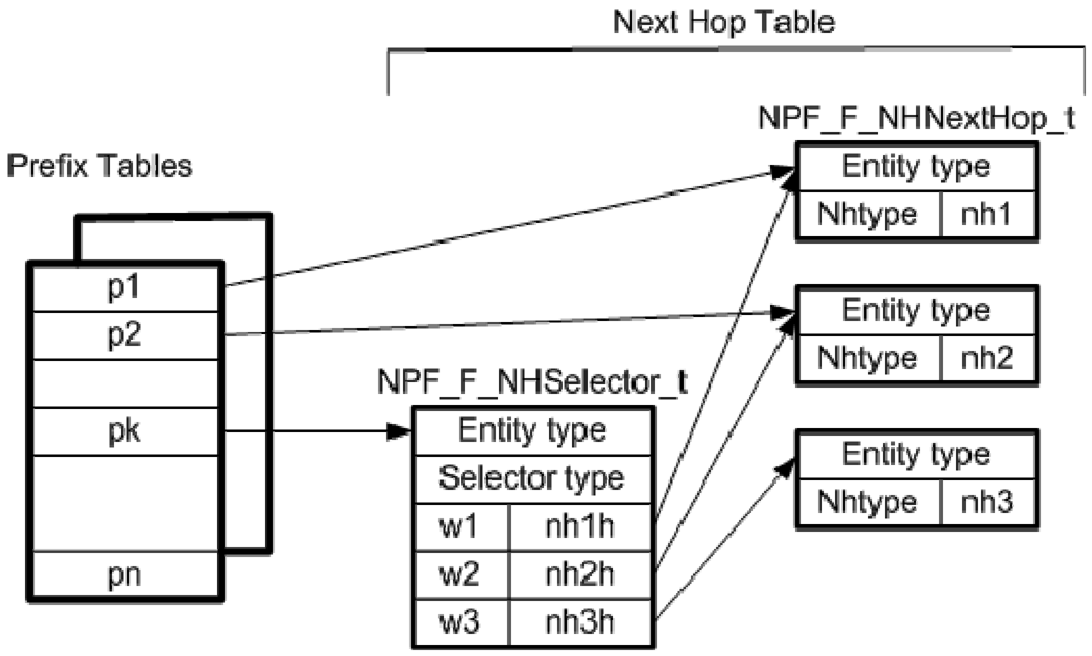

Figure 6 Two table approach

Both approaches are supported by the same API which takes a union of the two structures and differentiate between the two through an Entity type field.

Note that referring to an NPF_F_NHSelector_t from an NPF_F_NHSelector_t to form recursions is optional and not disallowed in this document. Implementations not allowing such recursions would return an error specifying that recursions are not supported.

4.1 Common LFB Data Types

4.1.1 Next Hop type

This enumerated type data structure is used to indicate whether a Next Hop is an IPv4, IPv6, or other type of next hop.

```
typedef enum {
      NPF_F_NHTYPE_RESERVED   = 0,
      NPF_F_NHTYPE_IPV4       = 1,
      NPF_F_NHTYPE_IPV6       = 2,
      NPF_F_NHTYPE_NHLFE      = 3
} NPF_F_NHType_t;
```

4.1.2 IPv4 Next Hop

An IPv4 Next hop contains an IPv4 address of type NPF_IPv4Address_t (refer to section 4.1.7).

4.1.3 IPv6 Next Hop

An IPv6 Next hop contains an IPv6 address of type NPF_IPv6Address_t (refer to section 4.1.7).

4.1.4 MPLS Next Hop

An MPLS Next hop data structure depicts an NHLFE and is defined as follows:

```
/*
 * Need to use the NHLFE entry id from NHLFE LFB (typedef)
 */
   typedef NPF_uint32_t NPF_F_NHLFE_Entryid_t;    /* This type should be
                                                *defined in the NHLFE LFB
                                                */
```

4.1.5 Next Hop Flags

The following typedef describes the flags used in the Next Hop Structure.

```
typedef NPF_uint32_t NPF_F_NHFlags_t;

#define  NPF_F_NHFLAGS_STATS_ON     0x1;  /*enables stats on the next hop*/
#define  NPF_F_NHFLAGS_COPYTOCP     0x2;  /*delivers a copy to CP*/
```

The NPF_F_NHFLAGS_COPYTOCP flag is used to deliver a copy of the packet to CP. This functionality can be used, as an example, for legal intercept.

4.1.6 Next Hop Options

The following enumeration describes the options used in the Next Hop Structure.

```
typedef enum {
   NPF_F_NHOPTION_BASIC       = 0,  /*use IP address in Next Hop struct*/
   NPF_F_NHOPTION_DIRECT      = 1,  /*use dest. IP in pkt*/
   NPF_F_NHOPTION_DROP        = 2,  /*Black hole*/
   NPF_F_NHOPTION_TOCP        = 3   /*Send to CP*/
} NPF_F_NHOption_t;
```

In order to forward the packet, this LFB can use the IP address specified in the Next Hop Structure (NPF_F_NHOPTION_BASIC – forward to next hop) or use the IP address specified in the IP destination field of the packet's header (NPF_F_NHOPTION_DIRECT – forward to locally attached host).

Note that options to drop or to send to CP will force the packet to be forwarded on the exception output port.

4.1.7 Next Hop Structure

A Next hop data structure is defined as follows:

```
typedef struct {
      NPF_F_NHOption_t  option;
      NPF_F_NHFlags_t   flags;
      NPF_uint32_t      egressMTU;
      NPF_uint32_t      egressVif; /*egress vif id*/
      NPF_uint8_t       TTL;
      NPF_F_NHType_t    type;
      union {
            NPF_IPv4Address_t       IPv4NextHop;
            NPF_IPv6Address_t       IPv6NextHop;
            NPF_F_NHLFE_Entryid_t         NHLFENextHop;
      } u;
} NPF_F_NHNextHop_t;
```

4.1.8 Next Hop Statistics Structure

A Next hop statistics structure is defined as follows:

```
typedef struct {
   NPF_uint64_t   packetCount;
   NPF_uint64_t   byteCount;
} NPF_F_NHStatistics_t;
```

Note that the statistics query functionality can help in verifying the load balancing schema and in debugging forwarding errors.

4.1.9 Next Hop Selector policy type

This enumerated type data structure is used to indicate whether the Next Hop Selector policy is based on weights, dscp values or other.

```
typedef enum {
      NPF_F_NHSELECTORTYPE_NONE            = 0,
      NPF_F_NHSELECTORTYPE_WEIGHT          = 1,  /*Based on weight*/
      NPF_F_NHSELECTORTYPE_DSCP            = 2,  /*Based on DSCP*/
      NPF_F_NHSELECTORTYPE_ROUNDROBIN      = 3   /*Round Robin per packet*/
} NPF_F_NHSelectorType_t;
```

4.1.10 Next Hop Selector weight policy

A Next Hop selector weight policy (selector type NPF_F_NHSELECTORTYPE_WEIGHT) uses an integer value to assign the weight per next hop, refer to section 4.1.14.

The Next Hop selection algorithm takes the weights associated with each Next Hop and distributes the load proportionally to the weights assigned (a Next hop with weight 2 should have double the traffic (i.e. the number of flows) than a next hop with weight 1, relatively to the Next Hops grouped in the Selector structure). This selector type insures that packets belonging to the same flow (e.g. source ip, destination ip, source port, destination port) MUST flow through the same next hop in order to preserve packet ordering within each flow. The algorithm to support this type is hardware dependent and is beyond the scope of this document. For further reference, [RFC2991] and [RFC2992] define ways and issues on handling multipath next hop selection.

4.1.11 Next Hop Selector DSCP policy

A Next Hop selector DSCP policy (selector type NPF_F_NHSELECTORTYPE_DSCP) uses an 8 bit field to specify the dscp code point that is associated with this next hop, refer to section 4.1.14.
The Next hop selection should be based on the incoming packet's DSCP value (or a derived equivalent DSCP value represented in the incoming metadata (optional metadata) if the DSCP is

not present in the packet's header e.g. MPLS encapsulated packet). For example, if the DSCP associated with the Next Hop structure through the above policy is 6, then all packets with DSCP 6 should be forwarded to the associated next hop. An LLSP (LSP ingress node) is an example of such policy.

4.1.12 Next Hop Selector Round Robin policy

A Next Hop selector Round Robin policy (selector type NPF_F_NHSELECTORTYPE_ROUNDROBIN) has the weight variable associated with it. A selector of this type will select in a weighted round robin fashion a next hop, out of the associated next hops, for each packet passing through it. This selector type does not enforce packet ordering within each flow (in fact, packets within each flow are expected to be reordered if this selector type is used, and it is up to the user to deal with any packet reordering problems). For example, if the selector has three next hops {A, B, C} with weights {3, 2, 1} respectively, the selector would forward packets into next hops in the following manner: A, B, C, A, B, A, A, B, C, A, B, A, etc.
Note for all weights equal to 1, the selector forwards packets in a simple round robin fashion: A, B, C, A, B, C, etc.

4.1.13 Next Hop Entity Id

The following typedef defines the Next Hop Entity Id:

```
typedef NPF_uint32_t NPF_F_NHEntityId_t;
```

Note that the Entity Id is specified by the application and can be reused in different Next Hop tables; The Entity Id should be unique within a Next Hop table.

4.1.14 Next Hop Selector entry

A Next Hop Selector entry data structure is defined as follows:

```
typedef struct {
      NPF_F_NHEntityId_t      nextHopId;
      union {
            NPF_uint32_t      weightPolicy;
            NPF_uint8_t       DSCPPolicy;
      } u;
} NPF_F_NHSelectorEntry_t;
```

Note: The nextHopId field in the NPF_F_NHSelectorEntry_t can refer to an NPF_F_NHNextHop_t structure or an NPF_F_NHSelector_t entry since recursions are allowed although they might be not supported in some implementations (an error specifying so is returned in the appropriate function call).
The weighPolicy or the DSCPPolicy fields, in the above structure, define the weight or the dscp associated with this next hop, as mentioned in sections 4.1.10 and 4.1.11, respectively.

4.1.15 Next Hop Selector

A Next hop Selector entry data structure is defined as follows:

```
typedef struct {
      NPF_F_NHSelectorType_t        type;
      NPF_uint32_t                  numEntries;
      NPF_F_NHSelectorEntry_t       *nhSelectorEntriesArray;
      NPF_F_NHEntityId_t            defaultNextHopEntityId;
} NPF_F_NHSelector_t;
```

All entries that are not covered by the specified nhSelectorEntriesArray will be forwarded to the Next Hop specified by defaultNextHopEntityId. For example, if a DSCP selector type is used, and the DSCP value is not matched in any entry, the defaultNextHopEntityId is selected as the next hop.

4.1.16 Next Hop Entity type

This enumerated type data structure is used to indicate whether the Next Hop Entity is a Next Hop Selector or a Next Hop.

```
typedef enum {
      NPF_F_NHENTITY_NONE          = 0,
      NPF_F_NHENTITY_NHSELECTOR    = 1,
      NPF_F_NHENTITY_NEXTHOP       = 2
} NPF_F_NHEntityType_t;
```

4.1.17 Next Hop Entity

A Next Hop Entity data structure is defined as follows:

```
typedef struct {
      NPF_F_NHEntityType_t          type;
      union {
            NPF_F_NHNextHop_t       nextHop;
            NPF_F_NHSelector_t      nhSelector;
      } u;
} NPF_F_NHEntity_t;
```

4.1.18 Next Hop Entity with Id

The following structure combines a Next Hop Entity with a Next Hop Entity Id

```
typedef struct {
      NPF_F_NHEntityId_t      nhEntityId;
      NPF_F_NHEntity_t        nhEntity;
} NPF_F_NHEntityWithId_t;
```

4.1.19 Next Hop Entity Query Response

A Next Hop Entity Query Response data structure is defined as follows:

```
typedef struct {
      NPF_F_NHEntityId_t      nhEntityId;
      NPF_F_NHEntity_t        nhEntity;
} NPF_F_NHEntityQueryResp_t;
```

4.1.20 Next Hop Table Handle

A Next hop table is uniquely identified by a table handle which is defined as follows:

```
typedef NPF_uint32_t NPF_F_NHTableHandle_t;
```

4.1.21 Next Hop Table Id

A Next hop table is uniquely identified by a table id which is defined as follows:

```
typedef NPF_uint32_t NPF_F_NHTableId_t;
```

Note: The table Id is provided from the FAPI client, whereas the Table handle is provided by the FAPI implementation.

4.1.22 Next Hop Table Handle Id

A Next Hop Table Handle Id data structure is defined as follows:

```
typedef struct {
      NPF_F_NHTableId_t       nhTableId;
      NPF_F_NHTableHandle_t   tableHandle;
} NPF_F_NHTableHandleId_t;
```

4.1.23 Next Hop Tables Handles Query Response

A Next Hop Tables Handles Query Response data structure is defined as follows:

```
typedef struct {
      NPF_uint32_t                  numEntries;
      NPF_F_NHTableHandleId_t       *nhTableHandleIdEntries;
} NPF_F_NHTablesHandlesQueryResp_t;
```

4.1.24 Next Hop LFB Query Response

A Next Hop LFB Query Response data structure is defined as follows:

```
typedef struct {
      NPF_uint32_t            maxNumTables;
      NPF_uint32_t            maxTableSize;
} NPF_F_NHLFBQueryResp_t;
```

4.1.25 Exception Handling

An exception is an occurrence of an event that triggers forwarding the packet beyond its normal processing path. The following enumeration specifies the various exceptions:

```
typedef enum {
      NPF_F_NH_EXCEPTION_RESERVED               = 0,
      NPF_F_NH_EXCEPTION_ICMP                   = 1,
      NPF_F_NH_EXCEPTION_RPF_FAILED             = 2,
      NPF_F_NH_EXCEPTION_TTL_ZERO               = 3,
      NPF_F_NH_EXCEPTION_DROP                   = 4,
      NPF_F_NH_EXCEPTION_SENDTOCP               = 5
} NPF_F_NHException_t;
```

4.2 Data Structures for Completion Callbacks

This section describes the completion callback data structures.

4.2.1 Asynchronous Response

This structure type definition holds asynchronous response/return information provided by a Next Hop API callback function response.

```
typedef struct {
      NPF_F_NHReturnCode_t    returnCode;
      union {
            NPF_F_NHTableHandle_t               tableHandle;
            NPF_uint32_t                        tableSpaceRemaining;
            NPF_F_NHEntityId_t                  nhEntityId;
            NPF_F_NHEntityQueryResp_t           nhEntityQueryResp;
            NPF_F_NHStatistics_t                nhStatistics;
            NPF_F_NHTablesHandlesQueryResp_t    nhTablesHandlesIds;
            NPF_F_NHLFBQueryResp_t              nhLFBInfo;
      }u;
} NPF_F_NHAsyncResponse_t;
```

4.2.2 Callback Type

This enumerated type definition specifies the callback types.

```
typedef enum {
      NPF_F_NH_RESERVED                               = 0,
      NPF_F_NH_TABLE_HANDLE_CREATE                    = 1,
      NPF_F_NH_TABLE_FLUSH                            = 2,
      NPF_F_NH_TABLE_HANDLE_DELETE                    = 3,
      NPF_F_NH_TABLE_ATTRIBUTE_QUERY                  = 4,
      NPF_F_NH_ENTITY_ADD                             = 5,
      NPF_F_NH_ENTITY_DELETE                          = 6,
      NPF_F_NH_ENTITY_MODIFY                          = 7,
      NPF_F_NH_ENTITY_FLAGS_MODIFY                    = 8,
      NPF_F_NH_ENTITY_OPTION_MODIFY                   = 9,
      NPF_F_NH_ENTITY_EGRESSMTU_MODIFY                = 10,
      NPF_F_NH_ENTITY_TTL_MODIFY                      = 11,
      NPF_F_NH_ENTITY_IPV4ADDR_MODIFY                 = 12,
      NPF_F_NH_ENTITY_IPV6ADDR_MODIFY                 = 13,
      NPF_F_NH_ENTITY_QUERY                           = 14,
      NPF_F_NH_STATISTICS_QUERY                       = 15,
      NPF_F_NH_STATISTICS_RESET                       = 16,
      NPF_F_NH_TABLESHANDLES_QUERY                    = 17,
      NPF_F_NH_LFB_QUERY                              = 18
} NPF_F_NHCallbackType_t;
```

4.2.3 Callback Data

This structure type definition holds the callback data information.

```
typedef struct {
      NPF_F_NHCallbackType_t  type;
      NPF_boolean_t           allOk;
      NPF_uint32_t            numResp;
      NPF_F_NHAsyncResponse_t *resp;
} NPF_F_NHCallbackData_t;
```

In the above structure, if all of the elements in the request issued by the original function call complete successfully and there is no additional response data to return, the callback will return an allOk value of NPF_TRUE, a numResp value of zero, and the array pointer will be null.
If not all of the responses are complete or if not all of the responses were successful or if there is additional response data to return, allOk will be NPF_FALSE, the numResp field will be greater than zero and the pointer to the resp array will be non-null. Failing elements will be determined by examining the return code in each array element.

The following table maps the type field to the union field used in the NPF_F_NHAsyncResponse_t structure. It also shows for each function call, the callback type returned.

Function	Callback Type	Callback Data
NPF_F_NHTableHandleCreate	NPF_F_NH_TABLE_HANDLE_CREATE	tableHandle
NPF_F_NHTableFlush	NPF_F_NH_TABLE_FLUSH	tableHandle
NPF_F_NHTableHandleDelete	NPF_F_NH_TABLE_HANDLE_DELETE	tableHandle
NPF_F_NHTableAttributeQuery	NPF_F_NH_TABLE_ATTRIBUTE_QUERY	tableSpaceRemaining
NPF_F_NHEntityAdd	NPF_F_NH_ENTITY_ADD	nhEntityId
NPF_F_NHEntityDelete	NPF_F_NH_ENTITY_DELETE	nhEntityId
NPF_F_NHEntityModify	NPF_F_NH_ENTITY_MODIFY	nhEntityId
NPF_F_NHEntityFlagsModify	NPF_F_NH_ FLAGS_MODIFY	nhEntityId
NPF_F_NHEntityOptionModify	NPF_F_NH_OPTION_MODIFY	nhEntityId
NPF_F_NHEntityEgressMTUModify	NPF_F_NH_ EGRESSMTU_MODIFY	nhEntityId
NPF_F_NHEntityTTLModify	NPF_F_NH_TTL_MODIFY	nhEntityId
NPF_F_NHEntityIPv4AddressModify	NPF_F_NH_ IPV4ADDR_MODIFY	nhEntityId
NPF_F_NHEntityIPv6AddressModify	NPF_F_NH_IPV6ADDR_MODIFY	nhEntityId
NPF_F_NHEntityQuery	NPF_F_NH_ENTITY_QUERY	nhEntityQueryResp
NPF_F_NHStatisticsQuery	NPF_F_NH_STATISTICS_QUERY	nhStatistics
NPF_F_NHStatisticsReset	NPF_F_NH_STATISTICS_RESET	nhEntityId
NPF_F_NHTablesHandleQuery	NPF_F_NH_TABLESHANDLES_QUERY	nhTablesHandlesIds
NPF_F_NHLFBQuery	NPF_F_NH_LFB_QUERY	nhLFBInfo

Table 1 NPF_F_NHCallbackType_t type mapping to NPF_F_NHAsyncResponse_t union type

4.3 Data Structures for Event Notifications

The following sections detail the information related to the Next Hop LFB events. When an event routine is invoked, one of the parameters will be a structure holding information about the occurred event.

4.3.1 Event Notification Types

The following enumeration indicates the type of the occurred event.

```
typedef enum {
   NPF_F_NH_TABLE_MISS_EVENT = 1,
   NPF_F_NH_ENTITY_MISS_EVENT = 2
} NPF_F_NHEvent_t;
```

The table miss event is triggered when the forwarding plane is unable to find a Next Hop table for a specific packet forwarded from the Prefix LFB. This event is optional.

The entity miss event is triggered when the forwarding plane is unable to find a Next Hop Entity within a Next Hop Table for a specific packet forwarded from the Prefix LFB. This event is optional.

4.3.2 Next Hop Table Miss Event

A Next Hop LFB Table Miss Event is defined as follows

```
typedef struct {
      NPF_F_NHTableId_t       tableId; /*Metadata received on input port*/
} NPF_F_NHTableMiss_Event_t;
```

4.3.3 Next Hop Entity Miss Event

A Next Hop LFB Entity Miss Event is defined as follows

```
typedef struct {
      NPF_F_NHTableId_t       tableId; /*Metadata received on input port*/
      NPF_F_NHEntityId_t      nhEntityId; /*Metadata received on input port*/
} NPF_F_NHEntityMiss_Event_t;
```

4.3.4 Event Notification Structures

This section describes the various events which MAY be supported.
It is important to note that if an implementation does not support any of these events, the implementation still needs to provide the event register and deregister functions to enable interoperability.
It is important to note also that care should be taken when generating events so that they don't overload the control plane. Rate limiting events is good practice in general, but is beyond the scope of this document.
The following structure defines all the possible event definitions for the Next Hop LFB. An event type field indicates which member of the union is relevant in the structure. The packetSize infers the size of the packet buffer that holds at least the IP packet header that caused the event.

```
typedef struct {
   NPF_F_NHEvent_t   type;
   union {
         NPF_F_NHTableMiss_Event_t        tableMiss;
         NPF_F_NHEntityMiss_Event_t       entityMiss;
   } u;

   /* Associated with an event is the IP packet
    */
   NPF_uint32_t   packetSize;
   NPF_uint8_t    *packetBuffer;
} NPF_F_NHEventInfo_t;
```

The following structure represents the events provided when the event notification routine is invoked:

```
typedef struct {
   NPF_uint32_t         numEvents;
   NPF_F_NHEventInfo_t  *eventArray;
} NPF_F_NHEventArray_t;
```

The following define bit masks used in the event registration function to allow the application to register for certain combination (all, some or none) of events.

```
/*
 * Definitions for Next Hop Events to be used in event mask
 */
#define NPF_F_NH_EVMASK_TABLE_MISS  (1<<1)
#define NPF_F_NH_EVMASK_ENTITY_MISS (1<<2)
```

4.4 Error Codes

4.4.1 Common NPF Error Codes

- **NPF_NO_ERROR --** This value MUST be returned when a function was successfully invoked. This value is also used in completion callbacks where it MUST be the only value used to signify success.
- **NPF_E_UNKNOWN --** An unknown error occurred in the implementation such that there is no error code defined that is more appropriate or informative.
- **NPF_E_BAD_CALLBACK_HANDLE --** A function was invoked with a callback handle that did not correspond to a valid NPF callback handle as returned by a registration function, or a callback handle was registered with a registration function belonging to a different API than the function call where the handle was passed in.
- **NPF_E_BAD_CALLBACK_FUNCTION --** A callback registration was invoked with a function pointer parameter that was invalid.
- **NPF_E_CALLBACK_ALREADY_REGISTERED --** A callback or event registration was invoked with a pair composed of a function pointer and a user context which was previously used for an identical registration.
- **NPF_E_FUNCTION_NOT_SUPPORTED --** This error value MUST be returned when an optional function call is not implemented by an implementation. This error value MUST NOT be returned by any required function call. This error value MUST be returned as the function return value (i.e. synchronously).
- **NPF_E_RESOURCE_EXISTS --** A duplicate request to create a resource was detected. No new resource was created.
- **NPF_E_RESOURCE_NONEXISTENT --** A duplicate request to destroy or free a resource was detected. The resource was previously destroyed or never existed.

4.4.2 LFB Specific Error Codes

The following type definition holds the error code returned in function callbacks.

typedef NPF_uint32_t NPF_F_NHReturnCode_t;

The following are error codes returned from an invocation of a function or in function callbacks:

```
#define NPF_F_NH_E_INSUFFICIENT_STORAGE        (NPF_F_NH_BASE_ERR+1)
#define NPF_F_NH_E_INVALID_ID                  (NPF_F_NH_BASE_ERR+2)
#define NPF_F_NH_E_BAD_FE_HANDLE               (NPF_F_NH_BASE_ERR+3)
```

```
#define NPF_F_NH_E_BAD_LFB_HANDLE              (NPF_F_NH_BASE_ERR+4)
#define NPF_F_NH_E_BAD_TABLE_HANDLE            (NPF_F_NH_BASE_ERR+5)
#define NPF_F_NH_E_NOT_AVAILABLE               (NPF_F_NH_BASE_ERR+6)
#define NPF_F_NH_E_ID_DOES_NOT_APPLY           (NPF_F_NH_BASE_ERR+7)
#define NPF_F_NH_E_BAD_EXCEPTION               (NPF_F_NH_BASE_ERR+8)
#define NPF_F_NH_E_RECURSION_NOT_SUPORTED (NPF_F_NH_BASE_ERR+9)
```

5 Functional APIs (FAPIs)

5.1 Required Functional APIs

5.1.1 Completion Callback Function

Syntax

```
typedef void (*NPF_F_NHCallBackFunc_t)(
   NPF_IN NPF_userContext_t         userContext,
   NPF_IN NPF_correlator_t          correlator,
   NPF_IN NPF_F_NHCallbackData_t    nhCallbackData);
```

Description

This callback function is used by the application to register an asynchronous response handling routine with the NPF FAPI Next Hop API implementation. This callback function is intended to be implemented by the application, and registered with the NPF FAPI Next Hop API implementation through NPF_F_NHRegister( ) function.

Input Parameters

- userContext – The context item that was supplied by the application when the completion callback function was registered.
- correlator – The correlator item that was supplied by the application when the FAPI Next Hop API function call was made. The correlator is used by the application mainly to distinguish between multiple invocations of the same function.
- nhCallbackData – Response information related to the FAPI Next Hop API function call. Contains information that are common among all functions, as well as information that are specific to a particular function. See NPF_F_NHCallbackData_t definition, section 4.2.3, for details.

Output Parameters

None.

Return Value

None.

Asynchronous Response

Not Applicable.

Notes

None.

5.1.2 Completion Callback Registration Function

Syntax

```
NPF_error_t NPF_F_NHRegister(
   NPF_IN NPF_userContext_t               userContext,
   NPF_IN NPF_F_NHCallBackFunc_t          nhCallbackFunc,
   NPF_OUT NPF_callbackHandle_t           *nhCallbackHandle);
```

Description

This function is used by an application to register its completion callback function for receiving asynchronous responses related to NPF FAPI NH API function calls. The application may register multiple callback functions using this function. The callback function is identified by the pair of userContext and nhCallbackFunc, and for each individual pair, a unique nhCallbackHandle will be assigned for future reference. Since the callback function is identified by both userContext and nhCallbackFunc, duplicate registration of same callback function with different userContext is allowed. Also, same userContext can be shared among different callback functions. Duplicate registration of the same userContext and nhCallbackFunc pair has no effect, and will output a handle that is already assigned to the pair, and will return NPF_E_CALLBACK_ALREADY_REGISTERED.

Note : NPF_F_NHRegister( ) is a synchronous function and has no completion callback associated with it.

Input Parameters

- userContext – A context item used for uniquely identifying the context of the application registering the completion callback function. The exact value will be provided back to the registered completion callback function as its 1st parameter when it is called. Application can assign any value to the userContext and the value is completely opaque to the NPF FAPI NH API implementation.
- nhCallbackFunc – The pointer to the completion callback function to be registered.

Output Parameters

- nhCallbackHandle – A unique identifier assigned for the registered userContext and nhCallbackFunc pair. This handle will be used by the application to specify which callback function to be called when invoking asynchronous NPF FAPI NH API functions. It will also be used when de-registering the userContext and nhCallbackFunc pair.

Return Values

- NPF_NO_ERROR – The registration completed successfully.
- NPF_E_BAD_CALLBACK_FUNCTION – nhCallbackFunc is NULL.
- NPF_E_CALLBACK_ALREADY_REGISTERED – No new registration was made since the userContext and nhCallbackFunc pair was already registered.
  Note: Whether this should be treated as an error or not is dependent on the application.

Asynchronous Response

Not Applicable.

Notes

None.

5.1.3 Completion Callback Deregistration

Syntax

```
NPF_error_t NPF_F_NHDeregister(
   NPF_IN NPF_callbackHandle_t      nhCallbackHandle);
```

Description

This function is used by an application to de-register a pair of user context and callback function.
Note: If there are any outstanding calls related to the de-registered callback function, the callback function may be called for those outstanding calls even after de-registration.
Note: NPF_F_NHDeregister( ) is a synchronous function and has no completion callback associated with it.

Input Parameters

- nhCallbackHandle – The unique identifier representing the pair of user context and callback function to be de-registered.

Output Parameters

None.

Return Values

- NPF_ NO_ERROR – The de-registration completed successfully.
- NPF_ E_BAD_CALLBACK_HANDLE – The API implementation does not recognize the callback handle. There is no effect to the registered callback functions.

Asynchronous Response

Not Applicable.

Notes

None.

5.1.4 Event Callback Function

Syntax

```
typedef void (*NPF_F_NHEventCallFunc_t) (
   NPF_IN NPF_userContext_t               userContext,
   NPF_OUT NPF_F_NHEventArray_t           data);
```

Description

This function is a registered event notification routine for handling Next Hop LFB Events.

Input Parameters

- userContext – A context item used for uniquely identifying the context of the application registering the completion callback function. The exact value will be provided back to the registered completion callback function as its 1st parameter when it is called. Application can assign any value to the userContext and the value is completely opaque to the NPF FAPI NH API implementation.
- data – A structure containing an array of event info structures and a count to indicate how many events are present.

Output Parameters

None.

Return Values

None.

Asynchronous Response

Not Applicable.

Notes

None.

5.1.5 Event Registration Function

Syntax

```
NPF_error_t NPF_F_NHEventRegister(
   NPF_IN NPF_userContext_t               userContext,
   NPF_IN NPF_F_NHEventCallFunc_t         nhEventCallFunc,
   NPF_IN NPF_eventMask_t                 eventMask,
   NPF_OUT NPF_callbackHandle_t           *nhEventCallHandle);
```

Description

This function is used by an application to register its event handling routine for receiving notifications of Next Hop LFB Events. The application may register multiple event handling routines using this function. The event handling routine is identified by the pair of userContext and nhEventCallFunc, and for each individual pair, a unique nhEventCallHandle will be assigned for future reference. Since the event handling routine is identified by both userContext and nhEventCallFunc, duplicate registration of same event handling routine with different userContext is allowed. Also, same userContext can be shared among different event handling routines. Duplicate registration of the same userContext and nhEventCallFunc pair has no effect, and will output a handle that is already assigned to the pair, and will return NPF_E_CALLBACK_ALREADY_REGISTERED.

Note : NPF_F_NHEventRegister ( ) is a synchronous function and has no completion callback associated with it.

Input Parameters

- userContext – A context item used for uniquely identifying the context of the application registering the completion callback function. The exact value will be provided back to the registered completion callback function as its 1st parameter when it is called. Application can assign any value to the userContext and the value is completely opaque to the NPF FAPI NH API implementation.
- eventMask – This is a bit mask of the Next Hop events. It allows the application to register for those selected events.
- nhEventCallFunc – The pointer to the event handling routine to be registered.

Output Parameters

- nhEventCallHandle – A unique identifier assigned for the registered userContext and nhEventCallFunc pair. This handle will be used by the application to specify which event handling routine to be called when invoking asynchronous NPF FAPI NH API functions. It will also be used when de-registering the userContext and nhEventCallFunc pair.

Return Values

- NPF_NO_ERROR – The registration completed successfully.
- NPF_E_BAD_CALLBACK_FUNCTION – nhEventCallFunc is NULL.
- NPF_E_CALLBACK_ALREADY_REGISTERED – No new registration was made since the userContext and nhEventCallFunc pair was already registered.
  Note: Whether this should be treated as an error or not is dependent on the application.

Asynchronous Response

Not Applicable.

Notes

None.

5.1.6 Event Deregistration Function

Syntax

```
NPF_error_t NPF_F_NHEventDeregister(
   NPF_IN NPF_callbackHandle_t      eventCallHandle);
```

Description

This function is used by an application to de-register a pair of user context and event Handler.
Note: If there are any outstanding calls related to the de-registered callback function, the callback function may be called for those outstanding calls even after de-registration.
Note: NPF_F_NHEventDeregister( ) is a synchronous function and has no completion callback associated with it.

Input Parameters

- eventCallHandle – The unique identifier representing the pair of user context and event Handler to be de-registered.

Output Parameters

None.

Return Values

- NPF_NO_ERROR – The de-registration completed successfully.
- NPF_E_BAD_CALLBACK_HANDLE – The API implementation does not recognize the event callback handle. There is no effect to the registered event Handler.

Asynchronous Response

Not Applicable.

Notes

None.

5.1.7 Next Hop Table Handle Create

Syntax

```
NPF_error_t NPF_F_NHTableHandleCreate (
   NPF_IN NPF_callbackHandle_t      callbackHandle,
   NPF_IN NPF_correlator_t          correlator,
   NPF_IN NPF_errorReporting_t      errorReporting,
   NPF_IN NPF_FEHandle_t            feHandle,
   NPF_IN NPF_BlockId_t             lfbHandle,
   NPF_IN NPF_F_NHTableId_t         nhTableId);
```

Description

This function creates a handle for a Next Hop Table with user specified Id. If the table Id is already in use, an error is returned stating so.

Input Parameters

- callbackHandle – The unique identifier provided to the application when the completion callback routine was registered.

- correlator – A unique application invocation value that will be supplied to the asynchronous completion callback routine.
- errorReporting – An indication of whether the application desires to receive an asynchronous completion callback for this API function call.
- nhTableId – A Next Hop Table Id generated by the application. Must be nonzero and different from Ids associated with Next Hop Tables previously created through this API.
- feHandle – The FE Handle returned by topology.
- lfbHandle – The Next Hop LFB Block Handle.

Output Parameters

None

Return Values

- NPF_NO_ERROR – The operation is in progress.
- NPF_E_UNKNOWN - The table handle creation did not execute due to unknown problems.
- NPF_E_BAD_CALLBACK_HANDLE – The callback handle is not valid.
- NPF_E_RESOURCE_EXISTS – The Next Hop Table id specified already exists. No new handle will be created.

Asynchronous Response

An NPF_F_NHCallbackData_t structure will be returned with NPF_F_NH_TABLE_HANDLE_CREATE as type, and the table handle will be returned in the tableHandle field of the embedded NPF_F_NHAsyncResponse_t struct. The returnCode field in the embedded NPF_F_NHAsyncResponse_t structure will carry one of the following possible error codes:

- NPF_NO_ERROR – The operation succeeded, and the returned table handle is valid.
- NPF_E_RESOURCE_EXISTS – The Next Hop Table id specified already exists. No new handle will be created. The returned handle is the current handle associated with the specified table id.
- NPF_F_NH_E_INSUFFICIENT_STORAGE – The operation failed due to lack of resources.
- NPF_F_NH_E_BAD_FE_HANDLE – The FE handle is not valid.
- NPF_F_NH_E_BAD_LFB_HANDLE – The LFB handle is not valid.

Notes

None.

5.1.8 Next Hop Table Flush

Syntax

```
NPF_error_t NPF_F_NHTableFlush (
    NPF_IN NPF_callbackHandle_t      callbackHandle,
    NPF_IN NPF_correlator_t          correlator,
    NPF_IN NPF_errorReporting_t      errorReporting,
    NPF_IN NPF_FEHandle_t            feHandle,
    NPF_IN NPF_BlockId_t             lfbHandle,
    NPF_IN NPF_F_NHTableHandle_t     nhTableHandle);
```

Description

This function removes all Next Hop entities from the Next Hop Table specified by nhTableHandle.

Input Parameters

- callbackHandle – The unique identifier provided to the application when the completion callback routine was registered.
- correlator – A unique application invocation value that will be supplied to the asynchronous completion callback routine.
- errorReporting – An indication of whether the application desires to receive an asynchronous completion callback for this API function call.
- nhTableHandle – The Next Hop Table Handle designating the table to be flushed.
- feHandle – The FE Handle returned by topology.
- lfbHandle – The Next Hop LFB Block Handle.

Output Parameters

None

Return Values

- NPF_NO_ERROR – The operation is in progress.
- NPF_E_UNKNOWN - The table flush operation did not execute due to unknown problems.
- NPF_F_NH_E_BAD_CALLBACK_HANDLE – The callback handle is not valid.

Asynchronous Response

An NPF_F_NHCallbackData_t structure will be returned with NPF_F_NH_TABLE_FLUSH as type, and the table handle will be returned in the tableHandle field of the embedded NPF_F_NHAsyncResponse_t struct. The returnCode field in the embedded NPF_F_NHAsyncResponse_t structure will carry one of the following possible error codes:

- NPF_NO_ERROR – The operation succeeded, the returned table handle is valid, and the table flushed.
- NPF_F_NH_E_BAD_TABLE_HANDLE – The table handle passed is invalid.
- NPF_E_UNKNOWN – Could not flush the table due to an unknown error.
- NPF_F_NH_E_BAD_FE_HANDLE – The FE handle is not valid.
- NPF_F_NH_E_BAD_LFB_HANDLE – The LFB handle is not valid.

Notes

None.

5.1.9 Next Hop Table Handle Delete

Syntax

```
NPF_error_t NPF_F_NHTableHandleDelete (
   NPF_IN NPF_callbackHandle_t      callbackHandle,
   NPF_IN NPF_correlator_t          correlator,
   NPF_IN NPF_errorReporting_t      errorReporting,
   NPF_IN NPF_FEHandle_t            feHandle,
   NPF_IN NPF_BlockId_t             lfbHandle,
   NPF_IN NPF_F_NHTableHandle_t     nhTableHandle);
```

Description

This function removes the Next Hop Table identified by nhTableHandle.

Input Parameters

- callbackHandle – The unique identifier provided to the application when the completion callback routine was registered.
- correlator – A unique application invocation value that will be supplied to the asynchronous completion callback routine.
- errorReporting – An indication of whether the application desires to receive an asynchronous completion callback for this API function call.
- nhTableHandle – The Next Hop Table Handle designating the table to be deleted.
- feHandle – The FE Handle returned by topology.
- lfbHandle – The Next Hop LFB Block Handle.

Output Parameters

None

Return Values

- NPF_NO_ERROR – The operation is in progress.
- NPF_E_UNKNOWN - The table delete operation did not execute due to unknown problems.
- NPF_E_BAD_CALLBACK_HANDLE – The callback handle is not valid.

Asynchronous Response

An NPF_F_NHCallbackData_t structure will be returned with NPF_F_NH_TABLE_HANDLE_DELETE as type, and the table handle will be returned in the tableHandle field of the embedded NPF_F_NHAsyncResponse_t struct. The returnCode field in the embedded NPF_F_NHAsyncResponse_t structure will carry one of the following possible error codes:

- NPF_NO_ERROR – The operation succeeded, the returned table handle is no longer valid, and the table is deleted.
- NPF_F_NH_E_BAD_TABLE_HANDLE – The table handle passed is invalid.
- NPF_E_UNKNOWN – Could not delete the table due to an unknown error.
- NPF_F_NH_E_BAD_FE_HANDLE – The FE handle is not valid.
- NPF_F_NH_E_BAD_LFB_HANDLE – The LFB handle is not valid.

Notes

None.

5.1.10 Next Hop Entity Add

Syntax

```
NPF_error_t NPF_F_NHEntityAdd (
   NPF_IN NPF_callbackHandle_t      callbackHandle,
   NPF_IN NPF_correlator_t          correlator,
   NPF_IN NPF_errorReporting_t      errorReporting,
   NPF_IN NPF_FEHandle_t            feHandle,
   NPF_IN NPF_BlockId_t             lfbHandle,
   NPF_IN NPF_F_NHTableHandle_t     nhTableHandle,
   NPF_IN NPF_uint32_t              numEntries,
   NPF_IN NPF_F_NHEntityWithId_t    *nhEntityWithIdArray);
```

Description

This function creates one or more Next Hop Entities in the specified Next Hop Table and returns the Ids of the created entities associated with an error code in the callback structure. The nhEntityWithIdArray is an array of size numEntries holding the Entities and their corresponding Ids.
If a table entry already exists (the entity id is already in use), then the newly added entry with same id will replace the old entry.

Input Parameters

- callbackHandle – The unique identifier provided to the application when the completion callback routine was registered.
- correlator – A unique application invocation value that will be supplied to the asynchronous completion callback routine.
- errorReporting – An indication of whether the application desires to receive an asynchronous completion callback for this API function call.
- nhTableHandle – The Next Hop Table Handle designating the table where the entities are added.
- feHandle – The FE Handle returned by topology.
- lfbHandle – The Next Hop LFB Block Handle.

- numEntries – The number of elements in the nhEntityIdArray and nhEntityArray. Each of these arrays has the same number of elements and they are positionally related.
- nhEntityWithIdArray – Pointer to an array of Next Hop Entities and Identifiers as determined by the caller.

Output Parameters

None

Return Values

- NPF_NO_ERROR – The operation is in progress.
- NPF_E_UNKNOWN – The entries were not added due to unknown problems encountered while handling the input parameters.
- NPF_E_BAD_CALLBACK_HANDLE – The callback handle is not valid.

Asynchronous Response

There may be multiple asynchronous callbacks to this request. An NPF_F_NHCallbackData_t structure will be returned with NPF_F_NH_ENTITY_ADD as type, and the entity id will be returned in the nhEntityId field of the embedded NPF_F_NHAsyncResponse_t struct. The returnCode field in the embedded NPF_F_NHAsyncResponse_t structure will carry one of the following possible error codes:

- NPF_NO_ERROR – The operation succeeded, and the returned Entity Id is valid.
- NPF_F_NH_E_INSUFFICIENT_STORAGE – The operation failed due to lack of resources.
- NPF_F_NH_E_BAD_TABLE_HANDLE – The table handle passed is invalid.
- NPF_E_UNKNOWN – Could not add entry due to an unknown error.
- NPF_F_NH_E_BAD_FE_HANDLE – The FE handle is not valid.
- NPF_F_NH_E_BAD_LFB_HANDLE – The LFB handle is not valid.
- NPF_F_NH_E_RECURSION_NOT_SUPORTED – Next Hop Selector entries form a recursion that is not supported.

Notes

None.

5.1.11 Next Hop Entity Delete

Syntax

```
NPF_error_t NPF_F_NHEntityDelete (
   NPF_IN NPF_callbackHandle_t      callbackHandle,
   NPF_IN NPF_correlator_t          correlator,
   NPF_IN NPF_errorReporting_t      errorReporting,
   NPF_IN NPF_FEHandle_t            feHandle,
   NPF_IN NPF_BlockId_t             lfbHandle,
   NPF_IN NPF_F_NHTableHandle_t     nhTableHandle,
   NPF_IN NPF_uint32_t              numEntries,
   NPF_IN NPF_F_NHEntityId_t        *nhEntityIdArray);
```

Description

This function deletes one or more Next Hop Entities in the specified Next Hop Table. The nhEntityIdArray points to an array of Next Hop Entity Ids of size numEntries.
If a table entry does not exist (the entity id is not in use), the entry id is silently discarded.
If a Next Hop Entry referenced by the Prefix LFB does not exist, the Next Hop LFB MAY generate an NPF_F_NH_TABLE_MISS_EVENT.

Input Parameters

- callbackHandle – The unique identifier provided to the application when the completion callback routine was registered.
- correlator – A unique application invocation value that will be supplied to the asynchronous completion callback routine.
- errorReporting – An indication of whether the application desires to receive an asynchronous completion callback for this API function call.
- nhTableHandle – The Next Hop Table Handle designating the table where the entities are to be deleted.
- feHandle – The FE Handle returned by topology.
- lfbHandle – The Next Hop LFB Block Handle.
- numEntries – The number of elements in the nhEntityIdArray.
- nhEntityIdArray – Pointer to an array of Next Hop Identifiers as determined by the caller.

Output Parameters

None

Return Values

- NPF_NO_ERROR – The operation is in progress.
- NPF_E_UNKNOWN – The entries were not deleted due to unknown problems encountered while handling the input parameters.
- NPF_E_BAD_CALLBACK_HANDLE – The callback handle is not valid.

Asynchronous Response

There may be multiple asynchronous callbacks to this request. An NPF_F_NHCallbackData_t structure will be returned with NPF_F_NH_ENTITY_DELETE as type and the entity id will be returned in the nhEntityId field of the embedded NPF_F_NHAsyncResponse_t struct. The returnCode field in the embedded NPF_F_NHAsyncResponse_t structure will carry one of the following possible error codes:

- NPF_NO_ERROR – The operation succeeded, and the returned Entity Id is valid.
- NPF_F_NH_E_BAD_TABLE_HANDLE – The table handle passed is invalid.
- NPF_E_UNKNOWN – Could not delete entry due to an unknown error.

- NPF_F_NH_E_BAD_FE_HANDLE – The FE handle is not valid.
- NPF_F_NH_E_BAD_LFB_HANDLE – The LFB handle is not valid.

Notes

None.

5.1.12 Next Hop Entity Modify

Syntax

```
NPF_error_t NPF_F_NHEntityModify (
   NPF_IN NPF_callbackHandle_t      callbackHandle,
   NPF_IN NPF_correlator_t          correlator,
   NPF_IN NPF_errorReporting_t      errorReporting,
   NPF_IN NPF_FEHandle_t            feHandle,
   NPF_IN NPF_BlockId_t             lfbHandle,
   NPF_IN NPF_F_NHTableHandle_t     nhTableHandle,
   NPF_IN NPF_uint32_t              numEntries,
   NPF_IN NPF_F_NHEntityWithId_t    *nhEntityWithIdArray);
```

Description

This function modifies one or more Next Hop Entities in the specified Next Hop Table and returns the Ids of the modified entities associated with error codes. The entries in the array would replace the existing forwarding plane Next Hop Entities specified by the entity id entry in nhEntityWithIdArray.
If a table entry does not exist (the entity id is not in use), an NPF_F_NH_E_INVALID_ID error is assigned in the returnCode field of the NPF_F_NHAsyncResponse_t structure.

Input Parameters

- callbackHandle – The unique identifier provided to the application when the completion callback routine was registered.
- correlator – A unique application invocation value that will be supplied to the asynchronous completion callback routine.
- errorReporting – An indication of whether the application desires to receive an asynchronous completion callback for this API function call.
- nhTableHandle – The Next Hop Table Handle designating the table where the entities are added.
- feHandle – The FE Handle returned by topology.
- lfbHandle – The Next Hop LFB Block Handle.
- numEntries – The number of elements in the nhEntityIdArray and nhEntityArray. Each of these arrays has the same number of elements and they are positionally related.
- nhEntityWithIdArray – Pointer to an array of Next Hop Entities and Identifiers as determined by the caller.

Output Parameters

None

Return Values

- NPF_NO_ERROR – The operation is in progress.
- NPF_E_UNKNOWN – The entries were not added due to unknown problems encountered while handling the input parameters.
- NPF_E_BAD_CALLBACK_HANDLE – The callback handle is not valid.

Asynchronous Response

There may be multiple asynchronous callbacks to this request. An NPF_F_NHCallbackData_t structure will be returned with NPF_F_NH_ENTITY_MODIFY as type and the entity id will be returned in the nhEntityId field of the embedded NPF_F_NHAsyncResponse_t struct. The returnCode field in the embedded NPF_F_NHAsyncResponse_t structure will carry one of the following possible error codes:

- NPF_NO_ERROR – The operation succeeded, and the returned Entity Id is valid.
- NPF_F_NH_E_INSUFFICIENT_STORAGE – The operation failed due to lack of resources.
- NPF_F_NH_E_BAD_TABLE_HANDLE – The table handle passed is invalid.
- NPF_E_UNKNOWN – Could not add entry due to an unknown error.
- NPF_F_NH_E_BAD_FE_HANDLE – The FE handle is not valid.
- NPF_F_NH_E_BAD_LFB_HANDLE – The LFB handle is not valid.
- NPF_F_NH_E_ID_DOES_NOT_APPLY – Function call doesn't apply on the passed id.
- NPF_F_NH_E_RECURSION_NOT_SUPORTED – Next Hop Selector entries form a recursion that is not supported.

Notes

None.

5.1.13 Next Hop Entity Flags Modify

Syntax

```
NPF_error_t NPF_F_NHEntityFlagsModify (
   NPF_IN NPF_callbackHandle_t      callbackHandle,
   NPF_IN NPF_correlator_t          correlator,
   NPF_IN NPF_errorReporting_t      errorReporting,
   NPF_IN NPF_FEHandle_t            feHandle,
   NPF_IN NPF_BlockId_t             lfbHandle,
   NPF_IN NPF_F_NHTableHandle_t     nhTableHandle,
   NPF_IN NPF_F_NHEntityId_t        nhEntityId,
   NPF_IN NPF_F_NHFlags_t           newFlags);
```

Description

This function modifies the flags field of the Next Hop Entity specified by the nhEntityId parameter. The newFlags field would replace the existing forwarding plane Next Hop Entity flags.

If a table entry does not exist (the entity id is not in use), an NPF_F_NH_E_INVALID_ID error is assigned in the returnCode field of the NPF_F_NHAsyncResponse_t structure.

Input Parameters

- callbackHandle – The unique identifier provided to the application when the completion callback routine was registered.
- correlator – A unique application invocation value that will be supplied to the asynchronous completion callback routine.
- errorReporting – An indication of whether the application desires to receive an asynchronous completion callback for this API function call.
- nhTableHandle – The Next Hop Table Handle designating the table where the entities are added.
- feHandle – The FE Handle returned by topology.
- lfbHandle – The Next Hop LFB Block Handle.
- nhEntityId – The Next Hop Entity Id identifying the Next Hop Entity.
- newFlags – The new flags to replace the forwarding plane flags in the identified Next Hop.

Output Parameters

None

Return Values

- NPF_NO_ERROR – The operation is in progress.
- NPF_E_UNKNOWN – The entries were not modified due to unknown problems encountered while handling the input parameters.
- NPF_E_BAD_CALLBACK_HANDLE – The callback handle is not valid.

Asynchronous Response

An NPF_F_NHCallbackData_t structure will be returned with NPF_F_NH_FLAGS_MODIFY as type, and the entity id will be returned in the nhEntityId field of the embedded NPF_F_NHAsyncResponse_t struct. The returnCode field in the embedded NPF_F_NHAsyncResponse_t structure will carry one of the following possible error codes:

- NPF_NO_ERROR – The operation succeeded, and the returned Entity Id is valid.
- NPF_F_NH_E_BAD_TABLE_HANDLE – The table handle passed is invalid.
- NPF_E_UNKNOWN – Could not modify entry due to an unknown error.
- NPF_F_NH_E_BAD_FE_HANDLE – The FE handle is not valid.
- NPF_F_NH_E_BAD_LFB_HANDLE – The LFB handle is not valid.
- NPF_F_NH_E_ID_DOES_NOT_APPLY – Function call doesn’t apply on the passed id.

Notes

None.

5.1.14 Next Hop Entity Option Modify

Syntax

```
NPF_error_t NPF_F_NHEntityOptionModify (
   NPF_IN NPF_callbackHandle_t      callbackHandle,
   NPF_IN NPF_correlator_t          correlator,
   NPF_IN NPF_errorReporting_t      errorReporting,
   NPF_IN NPF_FEHandle_t            feHandle,
   NPF_IN NPF_BlockId_t             lfbHandle,
   NPF_IN NPF_F_NHTableHandle_t     nhTableHandle,
   NPF_IN NPF_F_NHEntityId_t        nhEntityId,
   NPF_IN NPF_F_NHOption_t          newOption);
```

Description

This function modifies the option field of the Next Hop Entity specified by the nhEntityId parameter. The newOption field would replace the existing forwarding plane Next Hop Entity option.
If a table entry does not exist (the entity id is not in use), an NPF_F_NH_E_INVALID_ID error is assigned in the returnCode field of the NPF_F_NHAsyncResponse_t structure.

Input Parameters

- callbackHandle – The unique identifier provided to the application when the completion callback routine was registered.
- correlator – A unique application invocation value that will be supplied to the asynchronous completion callback routine.
- errorReporting – An indication of whether the application desires to receive an asynchronous completion callback for this API function call.
- nhTableHandle – The Next Hop Table Handle designating the table where the entities are added.
- feHandle – The FE Handle returned by topology.
- lfbHandle – The Next Hop LFB Block Handle.
- nhEntityId – The Next Hop Entity Id identifying the Next Hop Entity.
- newOption – The new option to replace the forwarding plane option in the identified Next Hop.

Output Parameters

None

Return Values

- NPF_NO_ERROR – The operation is in progress.
- NPF_E_UNKNOWN – The entries were not modified due to unknown problems encountered while handling the input parameters.
- NPF_E_BAD_CALLBACK_HANDLE – The callback handle is not valid.

Asynchronous Response

An NPF_F_NHCallbackData_t structure will be returned with NPF_F_NH_ENTITY_OPTION_MODIFY as type, and the entity id will be returned in the nhEntityId field of the embedded NPF_F_NHAsyncResponse_t struct. The returnCode field in the embedded NPF_F_NHAsyncResponse_t structure will carry one of the following possible error codes:

- NPF_NO_ERROR – The operation succeeded, and the returned Entity Id is valid.
- NPF_F_NH_E_BAD_TABLE_HANDLE – The table handle passed is invalid.
- NPF_E_UNKNOWN – Could not modify entry due to an unknown error.
- NPF_F_NH_E_BAD_FE_HANDLE – The FE handle is not valid.
- NPF_F_NH_E_BAD_LFB_HANDLE – The LFB handle is not valid.
- NPF_F_NH_E_ID_DOES_NOT_APPLY – Function call doesn't apply on the passed id.

Notes

None.

5.1.15 Next Hop Entity egressMTU Modify

Syntax

```
NPF_error_t NPF_F_NHEntityEgressMTUModify (
   NPF_IN NPF_callbackHandle_t      callbackHandle,
   NPF_IN NPF_correlator_t          correlator,
   NPF_IN NPF_errorReporting_t      errorReporting,
   NPF_IN NPF_FEHandle_t            feHandle,
   NPF_IN NPF_BlockId_t             lfbHandle,
   NPF_IN NPF_F_NHTableHandle_t     nhTableHandle,
   NPF_IN NPF_F_NHEntityId_t        nhEntityId,
   NPF_IN NPF_uint32_t              newEgressMTU);
```

Description

This function modifies the egressMTU field of the Next Hop Entity specified by the nhEntityId parameter. The newEgressMTU field would replace the existing forwarding plane Next Hop Entity egress MTU.
If a table entry does not exist (the entity id is not in use), an NPF_F_NH_E_INVALID_ID error is assigned in the returnCode field of the NPF_F_NHAsyncResponse_t structure.

Input Parameters

- callbackHandle – The unique identifier provided to the application when the completion callback routine was registered.
- correlator – A unique application invocation value that will be supplied to the asynchronous completion callback routine.
- errorReporting – An indication of whether the application desires to receive an asynchronous completion callback for this API function call.

- nhTableHandle – The Next Hop Table Handle designating the table where the entities are added.
- feHandle – The FE Handle returned by topology.
- lfbHandle – The Next Hop LFB Block Handle.
- nhEntityId – The Next Hop Entity Id identifying the Next Hop Entity.
- newEgressMTU – The new egress MTU to replace the forwarding plane egress MTU in the identified Next Hop.

Output Parameters

None

Return Values

- NPF_NO_ERROR – The operation is in progress.
- NPF_E_UNKNOWN – The entries were not modified due to unknown problems encountered while handling the input parameters.
- NPF_E_BAD_CALLBACK_HANDLE – The callback handle is not valid.

Asynchronous Response

An NPF_F_NHCallbackData_t structure will be returned with NPF_F_NH_EGRESSMTU_MODIFY as type, and the entity id will be returned in the nhEntityId field of the embedded NPF_F_NHAsyncResponse_t struct. The returnCode field in the embedded NPF_F_NHAsyncResponse_t structure will carry one of the following possible error codes:

- NPF_NO_ERROR – The operation succeeded, and the returned Entity Id is valid.
- NPF_F_NH_E_BAD_TABLE_HANDLE – The table handle passed is invalid.
- NPF_E_UNKNOWN – Could not modify entry due to an unknown error.
- NPF_F_NH_E_BAD_FE_HANDLE – The FE handle is not valid.
- NPF_F_NH_E_BAD_LFB_HANDLE – The LFB handle is not valid.
- NPF_F_NH_E_ID_DOES_NOT_APPLY – Function call doesn't apply on the passed id.

Notes

None.

5.1.16 Next Hop Entity TTL Modify

Syntax

```
NPF_error_t NPF_F_NHEntityTTLModify (
   NPF_IN NPF_callbackHandle_t      callbackHandle,
   NPF_IN NPF_correlator_t          correlator,
   NPF_IN NPF_errorReporting_t      errorReporting,
   NPF_IN NPF_FEHandle_t            feHandle,
   NPF_IN NPF_BlockId_t             lfbHandle,
   NPF_IN NPF_F_NHTableHandle_t     nhTableHandle,
   NPF_IN NPF_F_NHEntityId_t        nhEntityId,
   NPF_IN NPF_uint8_t               newTTL);
```

Description

This function modifies the TTL field of the Next Hop Entity specified by the nhEntityId parameter. The newTTL field would replace the existing forwarding plane Next Hop Entity TTL. If a table entry does not exist (the entity id is not in use), an NPF_F_NH_E_INVALID_ID error is assigned in the returnCode field of the NPF_F_NHAsyncResponse_t structure.

Input Parameters

- callbackHandle – The unique identifier provided to the application when the completion callback routine was registered.
- correlator – A unique application invocation value that will be supplied to the asynchronous completion callback routine.
- errorReporting – An indication of whether the application desires to receive an asynchronous completion callback for this API function call.
- nhTableHandle – The Next Hop Table Handle designating the table where the entities are added.
- feHandle – The FE Handle returned by topology.
- lfbHandle – The Next Hop LFB Block Handle.
- nhEntityId – The Next Hop Entity Id identifying the Next Hop Entity.
- newTTL – The new TTL to replace the forwarding plane TTL in the identified Next Hop.

Output Parameters

None

Return Values

- NPF_NO_ERROR – The operation is in progress.
- NPF_E_UNKNOWN – The entries were not modified due to unknown problems encountered while handling the input parameters.
- NPF_E_BAD_CALLBACK_HANDLE – The callback handle is not valid.

Asynchronous Response

An NPF_F_NHCallbackData_t structure will be returned with NPF_F_NH_ENTITY_TTL_MODIFY as type, and the entity id will be returned in the nhEntityId field of the embedded NPF_F_NHAsyncResponse_t struct. The returnCode field in the embedded NPF_F_NHAsyncResponse_t structure will carry one of the following possible error codes:

- NPF_NO_ERROR – The operation succeeded, and the returned Entity Id is valid.
- NPF_F_NH_E_BAD_TABLE_HANDLE – The table handle passed is invalid.
- NPF_E_UNKNOWN – Could not modify entry due to an unknown error.
- NPF_F_NH_E_BAD_FE_HANDLE – The FE handle is not valid.

- NPF_F_NH_E_BAD_LFB_HANDLE – The LFB handle is not valid.
- NPF_F_NH_E_ID_DOES_NOT_APPLY – Function call doesn't apply on the passed id.

Notes

None.

5.1.17 Next Hop Entity IPv4 Address Modify

Syntax

```
NPF_error_t NPF_F_NHEntityIPv4AddressModify (
   NPF_IN NPF_callbackHandle_t      callbackHandle,
   NPF_IN NPF_correlator_t          correlator,
   NPF_IN NPF_errorReporting_t      errorReporting,
   NPF_IN NPF_FEHandle_t            feHandle,
   NPF_IN NPF_BlockId_t             lfbHandle,
   NPF_IN NPF_F_NHTableHandle_t     nhTableHandle,
   NPF_IN NPF_F_NHEntityId_t        nhEntityId,
   NPF_IN NPF_IPv4Address_t         newIPv4Address);
```

Description

This function modifies the IPv4 Next Hop Address field of the Next Hop Entity IPv4NextHop element of the union specified by the nhEntityId parameter. The newIPv4Address field would replace the existing forwarding plane Next Hop Entity IPv4 Next Hop Address.
If a table entry does not exist (the entity id is not in use), an NPF_F_NH_E_INVALID_ID error is assigned in the returnCode field of the NPF_F_NHAsyncResponse_t structure.

Input Parameters

- callbackHandle – The unique identifier provided to the application when the completion callback routine was registered.
- correlator – A unique application invocation value that will be supplied to the asynchronous completion callback routine.
- errorReporting – An indication of whether the application desires to receive an asynchronous completion callback for this API function call.
- nhTableHandle – The Next Hop Table Handle designating the table where the entities are added.
- feHandle – The FE Handle returned by topology.
- lfbHandle – The Next Hop LFB Block Handle.
- nhEntityId – The Next Hop Entity Id identifying the Next Hop Entity.
- newIPv4Address – The new IPv4 Address to replace the forwarding plane IPv4 Next Hop Address in the identified Next Hop.

Output Parameters

None

Return Values

- NPF_NO_ERROR – The operation is in progress.
- NPF_E_UNKNOWN – The entries were not modified due to unknown problems encountered while handling the input parameters.
- NPF_E_BAD_CALLBACK_HANDLE – The callback handle is not valid.

Asynchronous Response

An NPF_F_NHCallbackData_t structure will be returned with NPF_F_NH_ENTITY_IPV4ADDR_MODIFY as type, and the entity id will be returned in the nhEntityId field of the embedded NPF_F_NHAsyncResponse_t struct. The returnCode field in the embedded NPF_F_NHAsyncResponse_t structure will carry one of the following possible error codes:

- NPF_NO_ERROR – The operation succeeded, and the returned Entity Id is valid.
- NPF_F_NH_E_BAD_TABLE_HANDLE – The table handle passed is invalid.
- NPF_E_UNKNOWN – Could not modify entry due to an unknown error.
- NPF_F_NH_E_BAD_FE_HANDLE – The FE handle is not valid.
- NPF_F_NH_E_BAD_LFB_HANDLE – The LFB handle is not valid.
- NPF_F_NH_E_ID_DOES_NOT_APPLY – Function call doesn’t apply on the passed id.

Notes

None.

5.1.18 Next Hop Entity IPv6 Address Modify

Syntax

```
NPF_error_t NPF_F_NHEntityIPv6AddressModify (
   NPF_IN NPF_callbackHandle_t      callbackHandle,
   NPF_IN NPF_correlator_t          correlator,
   NPF_IN NPF_errorReporting_t      errorReporting,
   NPF_IN NPF_FEHandle_t            feHandle,
   NPF_IN NPF_BlockId_t             lfbHandle,
   NPF_IN NPF_F_NHTableHandle_t     nhTableHandle,
   NPF_IN NPF_F_NHEntityId_t        nhEntityId,
   NPF_IN NPF_IPv6Address_t         newIPv6Address);
```

Description

This function modifies the IPv4 Next Hop Address field of the Next Hop Entity IPv6NextHop element of the union specified by the nhEntityId parameter. The newIPv6Address field would replace the existing forwarding plane Next Hop Entity IPv6 Next Hop Address.
If a table entry does not exist (the entity id is not in use), an NPF_F_NH_E_INVALID_ID error is assigned in the returnCode field of the NPF_F_NHAsyncResponse_t structure.

Input Parameters

- callbackHandle – The unique identifier provided to the application when the completion callback routine was registered.
- correlator – A unique application invocation value that will be supplied to the asynchronous completion callback routine.
- errorReporting – An indication of whether the application desires to receive an asynchronous completion callback for this API function call.
- nhTableHandle – The Next Hop Table Handle designating the table where the entities are added.
- feHandle – The FE Handle returned by topology.
- lfbHandle – The Next Hop LFB Block Handle.
- nhEntityId – The Next Hop Entity Id identifying the Next Hop Entity.
- newIPv6Address – The new IPv6 Address to replace the forwarding plane IPv6 Next Hop Address in the identified Next Hop.

Output Parameters

None

Return Values

- NPF_NO_ERROR – The operation is in progress.
- NPF_E_UNKNOWN – The entries were not modified due to unknown problems encountered while handling the input parameters.
- NPF_E_BAD_CALLBACK_HANDLE – The callback handle is not valid.

Asynchronous Response

An NPF_F_NHCallbackData_t structure will be returned with NPF_F_NH_ENTITY_IPV6ADDR_MODIFY as type, and the entity id will be returned in the nhEntityId field of the embedded NPF_F_NHAsyncResponse_t struct. The returnCode field in the embedded NPF_F_NHAsyncResponse_t structure will carry one of the following possible error codes:

- NPF_NO_ERROR – The operation succeeded, and the returned Entity Id is valid.
- NPF_F_NH_E_BAD_TABLE_HANDLE – The table handle passed is invalid.
- NPF_E_UNKNOWN – Could not modify entry due to an unknown error.
- NPF_F_NH_E_BAD_FE_HANDLE – The FE handle is not valid.
- NPF_F_NH_E_BAD_LFB_HANDLE – The LFB handle is not valid.
- NPF_F_NH_E_ID_DOES_NOT_APPLY – Function call doesn’t apply on the passed id.

Notes

None.

5.2 Optional Functional APIs

5.2.1 Next Hop Table Attribute Query

Syntax

```
NPF_error_t NPF_F_NHTableAttributeQuery (
   NPF_IN NPF_callbackHandle_t      callbackHandle,
   NPF_IN NPF_correlator_t          correlator,
   NPF_IN NPF_errorReporting_t      errorReporting,
   NPF_IN NPF_FEHandle_t            feHandle,
   NPF_IN NPF_BlockId_t             lfbHandle,
   NPF_IN NPF_F_NHTableHandle_t     nhTableHandle);
```

Description

This function provides information about the specified Next Hop Table. Currently, the attributes returned are:

- An estimate of the how many free entries are in this table.

Input Parameters

- callbackHandle – The unique identifier provided to the application when the completion callback routine was registered.
- correlator – A unique application invocation value that will be supplied to the asynchronous completion callback routine.
- errorReporting – An indication of whether the application desires to receive an asynchronous completion callback for this API function call.
- nhTableHandle – The Next Hop table handle to be queried.
- feHandle – The FE Handle returned by topology.
- lfbHandle – The Next Hop LFB Block Handle.

Output Parameters

None

Return Values

- NPF_NO_ERROR – The operation is in progress.
- NPF_E_UNKNOWN - The table was not queried due to unknown problems.
- NPF_E_BAD_CALLBACK_HANDLE – The callback handle is not valid.
- NPF_E_FUNCTION_NOT_SUPPORTED – The attribute query capability is not supported by this implementation.

Asynchronous Response

An NPF_F_NHCallbackData_t structure will be returned with NPF_F_NH_TABLE_ATTRIBUTE_QUERY as type, and the table space remaining will be returned in the tableSpaceRemaining field of the embedded NPF_F_NHAsyncResponse_t struct. The returnCode field in the embedded NPF_F_NHAsyncResponse_t structure will carry one of the following possible error codes:

- NPF_NO_ERROR – The operation succeeded, and the number of free entries returned in the tableSpaceRemaining field.
- NPF_F_NH_E_BAD_TABLE_HANDLE – The Table handle is not valid.
- NPF_E_FUNCTION_NOT_SUPPORTED – The attribute query capability is not supported by this implementation.
- NPF_F_NH_E_BAD_FE_HANDLE – The FE handle is not valid.
- NPF_F_NH_E_BAD_LFB_HANDLE – The LFB handle is not valid.

Notes

The amount of free space returned should be the worst-case conditions value, so that the application can be assured that at least this many “Add” requests will succeed. In other words, the implementation SHOULD be conservative in what it returns.

5.2.2 Next Hop Entity Query

Syntax

```
NPF_error_t NPF_F_NHEntityQuery (
   NPF_IN NPF_callbackHandle_t      callbackHandle,
   NPF_IN NPF_correlator_t          correlator,
   NPF_IN NPF_errorReporting_t      errorReporting,
   NPF_IN NPF_FEHandle_t            feHandle,
   NPF_IN NPF_BlockId_t             lfbHandle,
   NPF_IN NPF_F_NHTableHandle_t     nhTableHandle,
   NPF_IN NPF_uint32_t              numEntries,
   NPF_IN NPF_F_NHEntityId_t        *nhEntityIdArray);
```

Description

This function queries one or more Next Hop Entities in the specified Next Hop Table. The nhEntityIdArray points to an array of Next Hop Entity Ids of size numEntries.
If the entries exist, the content of the entries are returned in the completion callback, otherwise an error of NPF_F_NH_E_INVALID_ID is assigned in the returnCode field of the NPF_F_NHAsyncResponse_t structure.

Note: if numEntries is 0 (nhEntityIdArray MUST be NULL), this function returns ALL Next Hop entities within that table.

Input Parameters

- callbackHandle – The unique identifier provided to the application when the completion callback routine was registered.
- correlator – A unique application invocation value that will be supplied to the asynchronous completion callback routine.
- errorReporting – An indication of whether the application desires to receive an asynchronous completion callback for this API function call.

- nhTableHandle – The Next Hop Table Handle designating the table where the entities are to be queried.
- feHandle – The FE Handle returned by topology.
- lfbHandle – The Next Hop LFB Block Handle.
- numEntries – The number of elements in the nhEntityIdArray.
- nhEntityIdArray – Pointer to an array of Next Hop Identifiers as determined by the caller.

Output Parameters

None

Return Values

- NPF_NO_ERROR – The operation is in progress.
- NPF_E_UNKNOWN – The entries were not queried due to unknown problems encountered while handling the input parameters.
- NPF_E_BAD_CALLBACK_HANDLE – The callback handle is not valid.
- NPF_E_FUNCTION_NOT_SUPPORTED – The entity query capability is not supported by this implementation.

Asynchronous Response

There may be multiple asynchronous callbacks to this request. An NPF_F_NHCallbackData_t structure will be returned with NPF_F_NH_ENTITY_QUERY as type, and the entity information will be returned in the nhEntityQueryResp field of the embedded NPF_F_NHAsyncResponse_t struct. The returnCode field in the embedded NPF_F_NHAsyncResponse_t structure will carry one of the following possible error codes:

- NPF_NO_ERROR – The operation succeeded, and the returned struct is valid.
- NPF_F_NH_E_INVALID_ID – The entity id is invalid.
- NPF_E_UNKNOWN – Could not delete entry due to an unknown error.
- NPF_F_NH_E_BAD_FE_HANDLE – The FE handle is not valid.
- NPF_F_NH_E_BAD_LFB_HANDLE – The LFB handle is not valid.

Notes

None.

5.2.3 Next Hop Statistics Query

Syntax

```
NPF_error_t NPF_F_NHStatisticsQuery (
   NPF_IN NPF_callbackHandle_t      callbackHandle,
   NPF_IN NPF_correlator_t          correlator,
   NPF_IN NPF_errorReporting_t      errorReporting,
   NPF_IN NPF_FEHandle_t            feHandle,
   NPF_IN NPF_BlockId_t             lfbHandle,
   NPF_IN NPF_F_NHTableHandle_t     nhTableHandle,
   NPF_IN NPF_uint32_t              numEntries,
   NPF_IN NPF_F_NHEntityId_t        *nhEntityIdArray);
```

Description

This function queries one or more Next Hop Entities Statistics in the specified Next Hop Table. The nhEntityIdArray points to an array of Next Hop Entity Ids of size numEntries.
If the entries statistics do not exist, an NPF_F_NH_E_NOT_AVAILABLE error is assigned in the returnCode field of the NPF_F_NHAsyncResponse_t structure.

Input Parameters

- callbackHandle – The unique identifier provided to the application when the completion callback routine was registered.
- correlator – A unique application invocation value that will be supplied to the asynchronous completion callback routine.
- errorReporting – An indication of whether the application desires to receive an asynchronous completion callback for this API function call.
- nhTableHandle – The Next Hop Table Handle designating the table where the entities are to be queried.
- feHandle – The FE Handle returned by topology.
- lfbHandle – The Next Hop LFB Block Handle.
- numEntries – The number of elements in the nhEntityIdArray.
- nhEntityIdArray – Pointer to an array of Next Hop Identifiers as determined by the caller.

Output Parameters

None

Return Values

- NPF_NO_ERROR – The operation is in progress.
- NPF_E_UNKNOWN – The entries were not queried due to unknown problems encountered while handling the input parameters.
- NPF_E_BAD_CALLBACK_HANDLE – The callback handle is not valid.
- NPF_E_FUNCTION_NOT_SUPPORTED – The statistics query capability is not supported by this implementation.

Asynchronous Response

There may be multiple asynchronous callbacks to this request. An NPF_F_NHCallbackData_t structure will be returned with NPF_F_NH_STATISTICS_QUERY as type, and the statistics will be returned in the nhStatistics field of the embedded NPF_F_NHAsyncResponse_t struct. The returnCode field in the embedded NPF_F_NHAsyncResponse_t structure will carry one of the following possible error codes:

- NPF_NO_ERROR – The operation succeeded, and the returned struct is valid.

- NPF_F_NH_E_INVALID_ID – The entity id is invalid.
- NPF_E_UNKNOWN – Could not delete entry due to an unknown error.
- NPF_F_NH_E_BAD_FE_HANDLE – The FE handle is not valid.
- NPF_F_NH_E_BAD_LFB_HANDLE – The LFB handle is not valid.
- NPF_F_NH_E_NOT_AVAILABLE – The stats for the queried entry are not available.

Notes

None.

5.2.4 Next Hop Statistics Reset

Syntax

```
NPF_error_t NPF_F_NHStatisticsReset (
   NPF_IN NPF_callbackHandle_t      callbackHandle,
   NPF_IN NPF_correlator_t          correlator,
   NPF_IN NPF_errorReporting_t      errorReporting,
   NPF_IN NPF_FEHandle_t            feHandle,
   NPF_IN NPF_BlockId_t             lfbHandle,
   NPF_IN NPF_F_NHTableHandle_t     nhTableHandle,
   NPF_IN NPF_uint32_t              numEntries,
   NPF_IN NPF_F_NHEntityId_t        *nhEntityIdArray);
```

Description

This function resets one or more Next Hop Entities Statistics in the specified Next Hop Table. The nhEntityIdArray points to an array of Next Hop Entity Ids of size numEntries.
If the entries statistics do not exist, an NPF_F_NH_E_NOT_AVAILABLE error is assigned in the returnCode field of the NPF_F_NHAsyncResponse_t structure.

Input Parameters

- callbackHandle – The unique identifier provided to the application when the completion callback routine was registered.
- correlator – A unique application invocation value that will be supplied to the asynchronous completion callback routine.
- errorReporting – An indication of whether the application desires to receive an asynchronous completion callback for this API function call.
- nhTableHandle – The Next Hop Table Handle designating the table where the entities are to be queried.
- feHandle – The FE Handle returned by topology.
- lfbHandle – The Next Hop LFB Block Handle.
- numEntries – The number of elements in the nhEntityIdArray.
- nhEntityIdArray – Pointer to an array of Next Hop Identifiers as determined by the caller.

Output Parameters

None

Return Values

- NPF_NO_ERROR – The operation is in progress.
- NPF_E_UNKNOWN – The entries were not reset due to unknown problems encountered while handling the input parameters.
- NPF_E_BAD_CALLBACK_HANDLE – The callback handle is not valid.
- NPF_E_FUNCTION_NOT_SUPPORTED – The statistics reset capability is not supported by this implementation.

Asynchronous Response

There may be multiple asynchronous callbacks to this request. An NPF_F_NHCallbackData_t structure will be returned with NPF_F_NH_STATISTICS_RESET as type, and the Next hop Id will be returned in the nhEntityId field of the embedded NPF_F_NHAsyncResponse_t struct. The returnCode field in the embedded NPF_F_NHAsyncResponse_t structure will carry one of the following possible error codes:

- NPF_NO_ERROR – The operation succeeded, and the returned struct is valid.
- NPF_F_NH_E_INVALID_ID – The entity id is invalid.
- NPF_E_UNKNOWN – Could not reset entry due to an unknown error.
- NPF_F_NH_E_BAD_FE_HANDLE – The FE handle is not valid.
- NPF_F_NH_E_BAD_LFB_HANDLE – The LFB handle is not valid.
- NPF_F_NH_E_NOT_AVAILABLE – The stats for the NH entries are not available.

Notes

None.

5.2.5 Next Hop Tables Handles Query

Syntax

```
NPF_error_t NPF_F_NHTablesHandleQuery (
   NPF_IN NPF_callbackHandle_t      callbackHandle,
   NPF_IN NPF_correlator_t          correlator,
   NPF_IN NPF_errorReporting_t      errorReporting,
   NPF_IN NPF_FEHandle_t            feHandle,
   NPF_IN NPF_BlockId_t             lfbHandle);
```

Description

This function returns All Next Hop table handles and their respective Ids within the specified FE and LFB.
If no Next Hop table exists an NPF_F_NH_E_NOT_AVAILABLE error is assigned in the returnCode field of the NPF_F_NHAsyncResponse_t structure.

Input Parameters

- callbackHandle – The unique identifier provided to the application when the completion callback routine was registered.
- correlator – A unique application invocation value that will be supplied to the asynchronous completion callback routine.
- errorReporting – An indication of whether the application desires to receive an asynchronous completion callback for this API function call.
- feHandle – The FE Handle returned by topology.
- lfbHandle – The Next Hop LFB Block Handle.

Output Parameters

None

Return Values

- NPF_NO_ERROR – The operation is in progress.
- NPF_E_UNKNOWN – The entries were not queried due to unknown problems encountered while handling the input parameters.
- NPF_E_BAD_CALLBACK_HANDLE – The callback handle is not valid.
- NPF_E_FUNCTION_NOT_SUPPORTED – The tables handles query capability is not supported by this implementation.

Asynchronous Response

There may be multiple asynchronous callbacks to this request. An NPF_F_NHCallbackData_t structure will be returned with NPF_F_NH_TABLESHANDLES_QUERY as type, and the handles will be returned in the nhTablesHandles field of the embedded NPF_F_NHAsyncResponse_t struct. The returnCode field in the embedded NPF_F_NHAsyncResponse_t structure will carry one of the following possible error codes:

- NPF_NO_ERROR – The operation succeeded, and the returned struct is valid.
- NPF_E_UNKNOWN – Could not query tables due to an unknown error.
- NPF_F_NH_E_BAD_FE_HANDLE – The FE handle is not valid.
- NPF_F_NH_E_BAD_LFB_HANDLE – The LFB handle is not valid.

Notes

None.

5.2.6 Next Hop LFB Query

Syntax

```
NPF_error_t NPF_F_NHLFBQuery (
   NPF_IN NPF_callbackHandle_t      callbackHandle,
   NPF_IN NPF_correlator_t          correlator,
   NPF_IN NPF_errorReporting_t      errorReporting,
   NPF_IN NPF_FEHandle_t            feHandle,
   NPF_IN NPF_BlockId_t            lfbHandle);
```

Description

This function returns the conservative estimate of the maximum number of Next Hop tables that could be created in this LFB, along with the conservative estimate of the maximum size (in number of entries) of a table.

Input Parameters

- callbackHandle – The unique identifier provided to the application when the completion callback routine was registered.
- correlator – A unique application invocation value that will be supplied to the asynchronous completion callback routine.
- errorReporting – An indication of whether the application desires to receive an asynchronous completion callback for this API function call.
- feHandle – The FE Handle returned by topology.
- lfbHandle – The Next Hop LFB Block Handle.

Output Parameters

None

Return Values

- NPF_NO_ERROR – The operation is in progress.
- NPF_E_UNKNOWN – The entries were not queried due to unknown problems encountered while handling the input parameters.
- NPF_E_BAD_CALLBACK_HANDLE – The callback handle is not valid.
- NPF_E_FUNCTION_NOT_SUPPORTED – The LFB query capability is not supported by this implementation.

Asynchronous Response

There could be only one asynchronous callback to this request. An NPF_F_NHCallbackData_t structure will be returned with NPF_F_NH_LFB_QUERY as type, and the number of tables will be returned in the nhLFBInfo field of the embedded NPF_F_NHAsyncResponse_t struct. The returnCode field in the embedded NPF_F_NHAsyncResponse_t structure will carry one of the following possible error codes:

- NPF_NO_ERROR – The operation succeeded, and the returned struct is valid.
- NPF_E_UNKNOWN – Could not query the LFB due to an unknown error.
- NPF_F_NH_E_BAD_FE_HANDLE – The FE handle is not valid.
- NPF_F_NH_E_BAD_LFB_HANDLE – The LFB handle is not valid.

Notes

None.

6 References

[FORCESREQ]	”Requirements for Separation of IP Control and Forwarding”, H. Khosravi, T. Anderson et al, July 2002. (http://www.ietf.org/internet-drafts/draft-ietf-forces-requirements-06.txt)
[FAPITOPO]	”Topology Manager Functional API”, Implementation Agreement, Network Processing Forum SWAPI Functional API TG, December 2004.
[IPV4SAPI]	“IPv4 Unicast Forwarding Service API” - Implementation Agreement Revision 2.0- Network Processing Forum SWAPI Foundations TG, June 2004
[IPV4FAPI]	“IPv4 Prefix LFB and Functional API” – Implementation Agreement – Network Processing Forum, February 2005
[RFC1812]	“Requirements for IP Version 4 Routers” – RFC 1812
[RFC2827]	“Network Ingress Filtering: Defeating Denial of Service Attacks which employ IP Source Address Spoofing” – RFC 2827
[RFC2991]	“Multipath issues in unicast and multicast next-hop selection” – RFC 2991
[RFC2992]	“Analysis of equal cost multi path algorithm” – RFC 2992
[RFC3270]	“Multi-Protocol Label Switching (MPLS) Support of Differentiated Services” – RFC 3270
[SWAPICON]	“Software API Conventions”, - Implementation Agreement - Network Processing Forum SWAPI Foundations TG, 2003

APPENDIX A HEADER FILE INFORMATION

```
typedef enum {
   NPF_F_NHTYPE_RESERVED      = 0,
   NPF_F_NHTYPE_IPV4          = 1,
   NPF_F_NHTYPE_IPV6          = 2,
   NPF_F_NHTYPE_NHLFE         = 3
} NPF_F_NHType_t;


/*
 * Need to use the NHLFE entry id from NHLFE LFB (typedef)
 */
typedef NPF_uint32_t NPF_F_NHLFE_Entryid_t ;/* This type should be defined in
                                            * the NHLFE LFB
                                            */

typedef NPF_uint32_t NPF_F_NHFlags_t;

#define  NPF_F_NHFLAGS_STATS_ON     0x1;  /*enables stats on the next hop*/
#define  NPF_F_NHFLAGS_COPYTOCP     0x2;  /*delivers a copy to CP*/

typedef enum {
   NPF_F_NHOPTION_BASIC       = 0,  /*use IP address in Next Hop struct*/
   NPF_F_NHOPTION_DIRECT      = 1,  /*use dest. IP in pkt*/
   NPF_F_NHOPTION_DROP        = 2,  /*Black hole*/
   NPF_F_NHOPTION_TOCP        = 3   /*Send to CP*/
} NPF_F_NHOption_t;

typedef struct {
   NPF_F_NHOption_t  option;
   NPF_F_NHFlags_t   flags;
   NPF_uint32_t      egressMTU;
   NPF_uint32_t      egressVif; /*egress vif id*/
   NPF_uint8_t       TTL;
   NPF_F_NHType_t    type;
   union {
      NPF_IPv4Address_t IPv4NextHop;
      NPF_IPv6Address_t       IPv6NextHop;
      NPF_F_NHLFEEntrid_t     NHLFENextHop;
   } u;
} NPF_F_NHNextHop_t;

typedef struct {
   NPF_uint64_t packetCount;
   NPF_uint64_t byteCount;
} NPF_F_NHStatistics_t;

typedef enum {
   NPF_F_NHSELECTORTYPE_NONE        = 0,
   NPF_F_NHSELECTORTYPE_WEIGHT      = 1,  /*Based on weight*/
   NPF_F_NHSELECTORTYPE_DSCP        = 2,  /*Based on DSCP*/
   NPF_F_NHSELECTORTYPE_ROUNDROBIN  = 3   /*Round Robin per packet*/
} NPF_F_NHSelectorType_t;
```

```
typedef NPF_uint32_t NPF_F_NHEntityId_t;

typedef struct {
   NPF_F_NHEntityId_t   nextHopId;
   union {
      NPF_uint32_t      weightPolicy;
      NPF_uint8_t       DSCPPolicy;
   } u;
} NPF_F_NHSelectorEntry_t;

typedef struct {
   NPF_F_NHSelectorType_t           type;
   NPF_uint32_t                     numEntries;
   NPF_F_NHSelectorEntry_t          *nhSelectorEntriesArray;
   NPF_F_NHEntityId_t               defaultNextHopEntityId;
} NPF_F_NHSelector_t;


typedef enum {
   NPF_F_NHENTITY_NONE              = 0,
   NPF_F_NHENTITY_NHSELECTOR        = 1,
   NPF_F_NHENTITY_NEXTHOP           = 2
} NPF_F_NHEntityType_t;

typedef struct {
   NPF_F_NHEntityType_t       type;
   union {
      NPF_F_NHNextHop_t       nextHop;
      NPF_F_NHSelector_t      nhSelector;
   } u;
} NPF_F_NHEntity_t;

typedef struct {
   NPF_F_NHEntityId_t   nhEntityId;
   NPF_F_NHEntity_t     nhEntity;
} NPF_F_NHEntityWithId_t;

typedef struct {
   NPF_F_NHEntityId_t   nhEntityId;
   NPF_F_NHEntity_t     nhEntity;
} NPF_F_NHEntityQueryResp_t;

typedef NPF_uint32_t NPF_F_NHTableHandle_t;

typedef NPF_uint32_t NPF_F_NHTableId_t;

typedef struct {
   NPF_F_NHTableId_t nhTableId;
   NPF_F_NHTableHandle_t      tableHandle;
} NPF_F_NHTableHandleId_t;

typedef struct {
   NPF_uint32_t               numEntries;
   NPF_F_NHTableHandleId_t    *nhTableHandleIdEntries;
} NPF_F_NHTablesHandlesQueryResp_t;
typedef struct {
      NPF_uint32_t            maxNumTables;
      NPF_uint32_t            maxTableSize;
} NPF_F_NHLFBQueryResp_t;


typedef enum {
   NPF_F_NH_EXCEPTION_RESERVED      = 0,
   NPF_F_NH_EXCEPTION_ICMP          = 1,
   NPF_F_NH_EXCEPTION_RPF_FAILED    = 2,
   NPF_F_NH_EXCEPTION_TTL_ZERO      = 3,
   NPF_F_NH_EXCEPTION_DROP          = 4,
   NPF_F_NH_EXCEPTION_SENDTOCP      = 5
} NPF_F_NHException_t;

typedef NPF_uint32_t NPF_F_NHReturnCode_t;

typedef struct {
   NPF_F_NHReturnCode_t returnCode;
   union {
      NPF_F_NHTableHandle_t               tableHandle;
      NPF_uint32_t                        tableSpaceRemaining;
      NPF_F_NHEntityId_t                  nhEntityId;
      NPF_F_NHEntityQueryResp_t           nhEntityQueryResp;
      NPF_F_NHStatistics_t                nhStatistics;
      NPF_F_NHTablesHandlesQueryResp_t    nhTablesHandlesIds;
      NPF_F_NHLFBQueryResp_t              nhLFBInfo;
   }u;
} NPF_F_NHAsyncResponse_t;

typedef enum {
      NPF_F_NH_RESERVED                                 = 0,
      NPF_F_NH_TABLE_HANDLE_CREATE                      = 1,
      NPF_F_NH_TABLE_FLUSH                              = 2,
      NPF_F_NH_TABLE_HANDLE_DELETE                      = 3,
      NPF_F_NH_TABLE_ATTRIBUTE_QUERY                    = 4,
      NPF_F_NH_ENTITY_ADD                               = 5,
      NPF_F_NH_ENTITY_DELETE                            = 6,
      NPF_F_NH_ENTITY_MODIFY                            = 7,
      NPF_F_NH_ENTITY_FLAGS_MODIFY                      = 8,
      NPF_F_NH_ENTITY_OPTION_MODIFY                     = 9,
      NPF_F_NH_ENTITY_EGRESSMTU_MODIFY                  = 10,
      NPF_F_NH_ENTITY_TTL_MODIFY                        = 11,
      NPF_F_NH_ENTITY_IPV4ADDR_MODIFY                   = 12,
      NPF_F_NH_ENTITY_IPV6ADDR_MODIFY                   = 13,
      NPF_F_NH_ENTITY_QUERY                             = 14,
      NPF_F_NH_STATISTICS_QUERY                         = 15,
      NPF_F_NH_STATISTICS_RESET                         = 16,
      NPF_F_NH_TABLESHANDLES_QUERY                      = 17,
      NPF_F_NH_LFB_QUERY                                = 18
} NPF_F_NHCallbackType_t;

typedef struct {
   NPF_F_NHCallbackType_t     type;
   NPF_boolean_t              allOk;
   NPF_uint32_t               numResp;
   NPF_F_NHAsyncResponse_t    *resp;
} NPF_F_NHCallbackData_t;
```

```
typedef enum {
   NPF_F_NH_TABLE_MISS_EVENT  = 1,
   NPF_F_NH_ENTITY_MISS_EVENT = 2
} NPF_F_NHEvent_t;

typedef struct {
      NPF_F_NHTableId_t       tableId; /*Metadata received on input port*/
} NPF_F_NHTableMiss_Event_t;

typedef struct {
      NPF_F_NHTableId_t       tableId; /*Metadata received on input port*/
      NPF_F_NHEntityId_t      nhEntityId; /*Metadata received on input port*/
} NPF_F_NHEntityMiss_Event_t;


typedef struct {
   NPF_F_NHEvent_t   type;
   union {
         NPF_F_NHTableMiss_Event_t        tableMiss;
         NPF_F_NHEntityMiss_Event_t       entityMiss;
   } u;

   /* Associated with an event is the IP packet
    */
   NPF_uint32_t   packetSize;
   NPF_uint8_t    *packetBuffer;
} NPF_F_NHEventInfo_t;

typedef struct {
   NPF_uint32_t         numEvents;
   NPF_F_NHEventInfo_t  *eventArray;
} NPF_F_NHEventArray_t;

#define NPF_F_NH_EVMASK_TABLE_MISS        (1<<1)
#define NPF_F_NH_EVMASK_ENTITY_MISS       (1<<2)

typedef void (*NPF_F_NHCallBackFunc_t)(
   NPF_IN NPF_userContext_t         userContext,
   NPF_IN NPF_correlator_t          correlator,
   NPF_IN NPF_F_NHCallbackData_t    nhCallbackData);

NPF_error_t NPF_F_NHRegister(
   NPF_IN NPF_userContext_t         userContext,
   NPF_IN NPF_F_NHCallBackFunc_t    nhCallbackFunc,
   NPF_OUT NPF_callbackHandle_t     *nhCallbackHandle);

NPF_error_t NPF_F_NHDeregister(
   NPF_IN NPF_callbackHandle_t      nhCallbackHandle);

typedef void (*NPF_F_NHEventCallFunc_t) (
   NPF_IN NPF_userContext_t   userContext,
   NPF_OUT NPF_F_NHEventArray_t     data);

NPF_error_t NPF_F_NHEventRegister(
   NPF_IN NPF_userContext_t         userContext,
   NPF_IN NPF_F_NHEventCallFunc_t   nhEventCallFunc,
   NPF_IN NPF_eventMask_t           eventMask,
   NPF_OUT NPF_callbackHandle_t     *nhEventCallHandle);

NPF_error_t NPF_F_NHEventDeregister(
   NPF_IN NPF_callbackHandle_t      eventCallHandle);

NPF_error_t NPF_F_NHTableHandleCreate (
   NPF_IN NPF_callbackHandle_t      callbackHandle,
   NPF_IN NPF_correlator_t          correlator,
   NPF_IN NPF_errorReporting_t      errorReporting,
   NPF_IN NPF_FEHandle_t            feHandle,
   NPF_IN NPF_BlockId_t             lfbHandle,
   NPF_IN NPF_F_NHTableId_t         nhTableId);

NPF_error_t NPF_F_NHTableFlush (
   NPF_IN NPF_callbackHandle_t      callbackHandle,
   NPF_IN NPF_correlator_t          correlator,
   NPF_IN NPF_errorReporting_t      errorReporting,
   NPF_IN NPF_FEHandle_t            feHandle,
   NPF_IN NPF_BlockId_t             lfbHandle,
   NPF_IN NPF_F_NHTableHandle_t     nhTableHandle);

NPF_error_t NPF_F_NHTableHandleDelete (
   NPF_IN NPF_callbackHandle_t      callbackHandle,
   NPF_IN NPF_correlator_t          correlator,
   NPF_IN NPF_errorReporting_t      errorReporting,
   NPF_IN NPF_FEHandle_t            feHandle,
   NPF_IN NPF_BlockId_t             lfbHandle,
   NPF_IN NPF_F_NHTableHandle_t     nhTableHandle);

NPF_error_t NPF_F_NHEntityAdd (
   NPF_IN NPF_callbackHandle_t      callbackHandle,
   NPF_IN NPF_correlator_t          correlator,
   NPF_IN NPF_errorReporting_t      errorReporting,
   NPF_IN NPF_FEHandle_t            feHandle,
   NPF_IN NPF_BlockId_t             lfbHandle,
   NPF_IN NPF_F_NHTableHandle_t     nhTableHandle,
   NPF_IN NPF_uint32_t              numEntries,
   NPF_IN NPF_F_NHEntityWithId_t    *nhEntityWithIdArray);

NPF_error_t NPF_F_NHEntityDelete (
   NPF_IN NPF_callbackHandle_t      callbackHandle,
   NPF_IN NPF_correlator_t          correlator,
   NPF_IN NPF_errorReporting_t      errorReporting,
   NPF_IN NPF_FEHandle_t            feHandle,
   NPF_IN NPF_BlockId_t             lfbHandle,
   NPF_IN NPF_F_NHTableHandle_t     nhTableHandle,
   NPF_IN NPF_uint32_t              numEntries,
   NPF_IN NPF_F_NHEntityId_t        *nhEntityIdArray);

NPF_error_t NPF_F_NHEntityModify (
   NPF_IN NPF_callbackHandle_t      callbackHandle,
   NPF_IN NPF_correlator_t          correlator,
   NPF_IN NPF_errorReporting_t      errorReporting,
   NPF_IN NPF_FEHandle_t            feHandle,
   NPF_IN NPF_BlockId_t             lfbHandle,
   NPF_IN NPF_F_NHTableHandle_t     nhTableHandle,
   NPF_IN NPF_uint32_t              numEntries,
   NPF_IN NPF_F_NHEntityWithId_t    *nhEntityWithIdArray);

NPF_error_t NPF_F_NHEntityFlagsModify (
   NPF_IN NPF_callbackHandle_t      callbackHandle,
   NPF_IN NPF_correlator_t          correlator,
   NPF_IN NPF_errorReporting_t      errorReporting,
   NPF_IN NPF_FEHandle_t            feHandle,
   NPF_IN NPF_BlockId_t             lfbHandle,
   NPF_IN NPF_F_NHTableHandle_t     nhTableHandle,
   NPF_IN NPF_F_NHEntityId_t        nhEntityId,
   NPF_IN NPF_F_NHFlags_t           newFlags);

NPF_error_t NPF_F_NHEntityOptionModify (
   NPF_IN NPF_callbackHandle_t      callbackHandle,
   NPF_IN NPF_correlator_t          correlator,
   NPF_IN NPF_errorReporting_t      errorReporting,
   NPF_IN NPF_FEHandle_t            feHandle,
   NPF_IN NPF_BlockId_t             lfbHandle,
   NPF_IN NPF_F_NHTableHandle_t     nhTableHandle,
   NPF_IN NPF_F_NHEntityId_t        nhEntityId,
   NPF_IN NPF_F_NHOption_t          newOption);

NPF_error_t NPF_F_NHEntityEgressMTUModify (
   NPF_IN NPF_callbackHandle_t      callbackHandle,
   NPF_IN NPF_correlator_t          correlator,
   NPF_IN NPF_errorReporting_t      errorReporting,
   NPF_IN NPF_FEHandle_t            feHandle,
   NPF_IN NPF_BlockId_t             lfbHandle,
   NPF_IN NPF_F_NHTableHandle_t     nhTableHandle,
   NPF_IN NPF_F_NHEntityId_t        nhEntityId,
   NPF_IN NPF_uint32_t              newEgressMTU);

NPF_error_t NPF_F_NHEntityTTLModify (
   NPF_IN NPF_callbackHandle_t      callbackHandle,
   NPF_IN NPF_correlator_t          correlator,
   NPF_IN NPF_errorReporting_t      errorReporting,
   NPF_IN NPF_FEHandle_t            feHandle,
   NPF_IN NPF_BlockId_t             lfbHandle,
   NPF_IN NPF_F_NHTableHandle_t     nhTableHandle,
   NPF_IN NPF_F_NHEntityId_t        nhEntityId,
   NPF_IN NPF_uint8_t               newTTL);

NPF_error_t NPF_F_NHEntityIPv4AddressModify (
   NPF_IN NPF_callbackHandle_t      callbackHandle,
   NPF_IN NPF_correlator_t          correlator,
   NPF_IN NPF_errorReporting_t      errorReporting,
   NPF_IN NPF_FEHandle_t            feHandle,
   NPF_IN NPF_BlockId_t             lfbHandle,
   NPF_IN NPF_F_NHTableHandle_t     nhTableHandle,
   NPF_IN NPF_F_NHEntityId_t        nhEntityId,
   NPF_IN NPF_IPv4Address_t         newIPv4Address);

NPF_error_t NPF_F_NHEntityIPv6AddressModify (
   NPF_IN NPF_callbackHandle_t      callbackHandle,
   NPF_IN NPF_correlator_t          correlator,
   NPF_IN NPF_errorReporting_t      errorReporting,
   NPF_IN NPF_FEHandle_t            feHandle,
   NPF_IN NPF_BlockId_t             lfbHandle,
   NPF_IN NPF_F_NHTableHandle_t     nhTableHandle,
   NPF_IN NPF_F_NHEntityId_t        nhEntityId,
   NPF_IN NPF_IPv6Address_t         newIPv6Address);

NPF_error_t NPF_F_NHTableAttributeQuery (
   NPF_IN NPF_callbackHandle_t      callbackHandle,
   NPF_IN NPF_correlator_t          correlator,
   NPF_IN NPF_errorReporting_t      errorReporting,
   NPF_IN NPF_FEHandle_t            feHandle,
   NPF_IN NPF_BlockId_t             lfbHandle,
   NPF_IN NPF_F_NHTableHandle_t     nhTableHandle);

NPF_error_t NPF_F_NHEntityQuery (
   NPF_IN NPF_callbackHandle_t      callbackHandle,
   NPF_IN NPF_correlator_t          correlator,
   NPF_IN NPF_errorReporting_t      errorReporting,
   NPF_IN NPF_FEHandle_t            feHandle,
   NPF_IN NPF_BlockId_t             lfbHandle,
   NPF_IN NPF_F_NHTableHandle_t     nhTableHandle,
   NPF_IN NPF_uint32_t              numEntries,
   NPF_IN NPF_F_NHEntityId_t        *nhEntityIdArray);

NPF_error_t NPF_F_NHStatisticsQuery (
   NPF_IN NPF_callbackHandle_t      callbackHandle,
   NPF_IN NPF_correlator_t          correlator,
   NPF_IN NPF_errorReporting_t      errorReporting,
   NPF_IN NPF_FEHandle_t            feHandle,
   NPF_IN NPF_BlockId_t             lfbHandle,
   NPF_IN NPF_F_NHTableHandle_t     nhTableHandle,
   NPF_IN NPF_uint32_t              numEntries,
   NPF_IN NPF_F_NHEntityId_t        *nhEntityIdArray);

NPF_error_t NPF_F_NHStatisticsReset (
   NPF_IN NPF_callbackHandle_t      callbackHandle,
   NPF_IN NPF_correlator_t          correlator,
   NPF_IN NPF_errorReporting_t      errorReporting,
   NPF_IN NPF_FEHandle_t            feHandle,
   NPF_IN NPF_BlockId_t             lfbHandle,
   NPF_IN NPF_F_NHTableHandle_t     nhTableHandle,
   NPF_IN NPF_uint32_t              numEntries,
   NPF_IN NPF_F_NHEntityId_t        *nhEntityIdArray);

NPF_error_t NPF_F_NHTablesHandleQuery (
   NPF_IN NPF_callbackHandle_t      callbackHandle,
   NPF_IN NPF_correlator_t          correlator,
   NPF_IN NPF_errorReporting_t      errorReporting,
   NPF_IN NPF_FEHandle_t            feHandle,
   NPF_IN NPF_BlockId_t             lfbHandle);

NPF_error_t NPF_F_NHLFBQuery (
   NPF_IN NPF_callbackHandle_t      callbackHandle,
   NPF_IN NPF_correlator_t          correlator,
   NPF_IN NPF_errorReporting_t      errorReporting,
   NPF_IN NPF_FEHandle_t            feHandle,
   NPF_IN NPF_BlockId_t             lfbHandle);
```

APPENDIX B FUNCTIONALITY NOT SUPPORTED IN IPV4 SAPI REV1.0

The following is a list of items that may need to be revised in the IPv4 SAPI IA Rev1.0 in order to support suggested functionality in the Next Hop LFB:

- This Next Hop LFB is based on the discrete model. The IPv4 SAPI states that the discrete model is optional.
- The notion of Next Hop Selector, firstly introduced in the MPLS SAPI IA, is not supported in the IPv4 SAPI. And thus, the flexibility of having different Next Hop Selector types is missing from the IPv4 SAPI IA Revision 1.0. Note that this document supports the MPLS SAPI Next Hop Selector Equivalent.
- The Next hop structures defined here and in the IPv4 SAPI are not aligned (TTL, ..).
- The IPv4 SAPI does not support optional counters per Next Hop entry.
- Optional functions NPF_F_NHStatisticsQuery(), NPF_F_NHStatisticsReset(), NPF_F_NHTablesHandleQuery(), are not supported in the IPv4 SAPI. Other individual next hop parameters set functions are also not supported.

APPENDIX C ACKNOWLEDGEMENTS

Working Group Chair: Alex Conta

Working Group Editor: John Renwick

Task Group Chair: Alistair Munro

The following individuals are acknowledged for their participation in the FAPI TG teleconferences, plenary meetings, mailing list, and/or for their NPF contributions used for the development of this Implementation Agreement. This list may not be all-inclusive since only names supplied by member companies for inclusion here will be listed. The NPF wishes to thank all active participants to this Implementation Agreement, whether listed here or not.

The list is in alphabetical order of last names:

Steven Blake, Modular Networks, Inc.
Gamil Cain, Intel
Ellen Deleganes, Intel
Reda Haddad, Ericsson (Document Editor)
Zsolt Harazsti, Modular Networks, Inc.
Hormuzd Khosravi, Intel
Vinoj Kumar, Agere Systems
David Maxwell, IDT
Alistair Munro, u4eatech
David Putzolu, Intel
John Renwick, Agere Systems

APPENDIX D LIST OF COMPANIES BELONGING TO NPF DURING APPROVAL PROCESS

Agere Systems	Hifn	PMC Sierra
Altera	IBM	Seaway Networks
AMCC	IDT	Sensory Networks
Analog Devices	Infineon Technologies AG	Sun Microsystems
Avici Systems	Intel	Teja Technologies
Cypress Semiconductor	IP Fabrics	TranSwitch
Enigma Semiconductor	IP Infusion	U4EA Group
Ericsson	Motorola	Wintegra
Erlang Technologies	Mercury Computer Systems	Xelerated
Flextronics	NetLogic	Xilinx
Freescale Semiconductor	Nokia	ZNYX Networks
HCL Technologies	NTT Electronics	