IPv4 Prefix LFB and Functional API Implementation Agreement

Feb 24, 2005
Revision 1.0

Editor(s):

Hormuzd Khosravi, Intel Corporation, hormuzd.m.khosravi@intel.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

1.0	01/12/2005	Created Rev 1.0 of the implementation agreement by taking the IPv4 Prefix FAPI document (npf2002.686.00) and making minor editorial corrections.

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 Prefix tables. It is responsible for LPM (longest prefix match) search on the packet to determine the matching Next Hop metadata for a particular prefix (destination IP address in normal case). It also performs checks to validate the prefix for the packet according to RFC 1812.
The FAPI Logical Function Block APIs are used to configure LFB resources and associate resources between LFBs. This document describes the Prefix LFB and its associated APIs.

2.1 Acronyms / Definitions

The following are acronyms used in this document.

API	**Application Programming Interface**
FAPI	**Functional API**
FIB	**Forwarding Information Base**
ICMP	**Internet Control Message Protocol**
LFB	**Logical Function Block**
LPM	**Longest Prefix Match**
RPF	**Reverse Path Forwarding**

2.2 Assumptions

There will be a separate Next Hop LFB defined which will consume some of the metadata generated by the Prefix LFB defined in this document.

2.3 Scope

This contribution concentrates on the details of the FAPI IPv4 Prefix LFB and APIs. A separate FAPI might handle any details of an IPv4 Forwarder LFB based on a single route table (unified) approach. A unified mode Service API implementation will need the control plane software to manage discrete prefix and next hop tables if it uses this LFB.

2.4 External Requirements and Dependencies

This API depends on the SWAPI Software Conventions [SWAPICON] Implementation Agreement. This API also depends on the FAPI Topology Discovery API Implementation Agreement. This API needs to be aligned with the requirements set by the ForCES WG [FORCESREQ] in the IETF.

3 IPv4 Prefix LFB

The IPv4 Prefix LFB performs some of the IPv4 header checks specified in RFC 1812. It then performs longest prefix match on the destination IP address of the packet and based on that adds a Next Hop ID and a Next Hop table ID metadata to the packet, which can be used by the IPv4 Next Hop LFB. This LFB will not handle any IP options (in particular, options related to looking up the next hop). This LFB maintains statistics for each Prefix table and provides an API to query those statistics. Upon a prefix lookup failure, this LFB will tag the packet with a special Next Hop ID and Next Hop table ID (it will not handle a prefix lookup failure).

In addition to managing prefixes, 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 performs LPM based on the source IP address of the packet, and generates the Next Hop ID and Next Hop table ID metadata based on the source IP. This packet is then forwarded on the RPF_OUT output port.

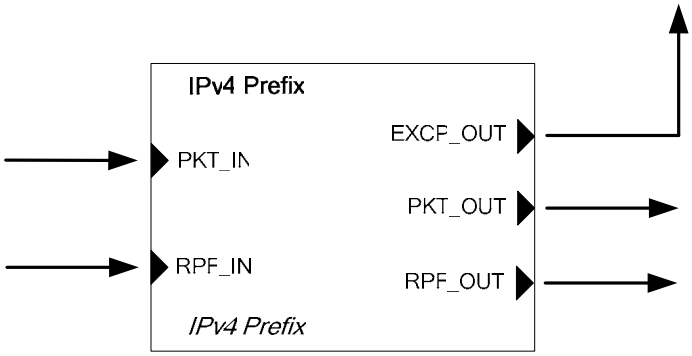

Figure 3-1 IPv4 Prefix LFB

3.1 LFB Input Ports

The IPv4 Prefix LFB contains input ports described in **Table 3-1**.

Table 3-1 IPv4 Prefix LFB Inputs

Input Port Symbolic Name	Description
PKT_IN	This is the normal packet input for IPv4 Prefix LFB.
RPF_IN	This is the input for packets on which RPF has to be performed.

3.1.1 Metadata Required

The input metadata contains a handle selecting the Prefix table, i.e. the prefix table handle.

3.2 LFB Output Ports

The IPv4 Prefix LFB contains output ports described in **Table 3-**2.

Table 3-2 IPv4 Prefix LFB Outputs

Input Port Symbolic Name	Description
PKT_OUT	This is the normal packet output from IPv4 Prefix LFB.
EXCP_OUT	The exception packets are sent to this output.
RPF_OUT	This is the output for packets that arrived on RPF_IN port, the Next Hop ID is generated based on source IP address.

3.2.1 Metadata Produced

The IPv4 Prefix LFB produces the Next Hop ID and the Next Hop table ID as metadata on the PKT_OUT port. This LFB will produce the Prefix Exception Id as metadata on the EXCP_OUT port.

3.3 Relationship with other LFBs

The IPv4 Prefix LFB works in conjunction with other LFBs such as Next Hop LFB and Packet Handler or Director LFB as shown in the Figure 2-2.

The Packet Handler or Director LFB handles the delivery of local or exception packets to the local/remote protocol stacks. The Next Hop LFB provides an API to manage, add and delete entries in the Next Hop Table. It assigns Layer 2 related metadata to packets.

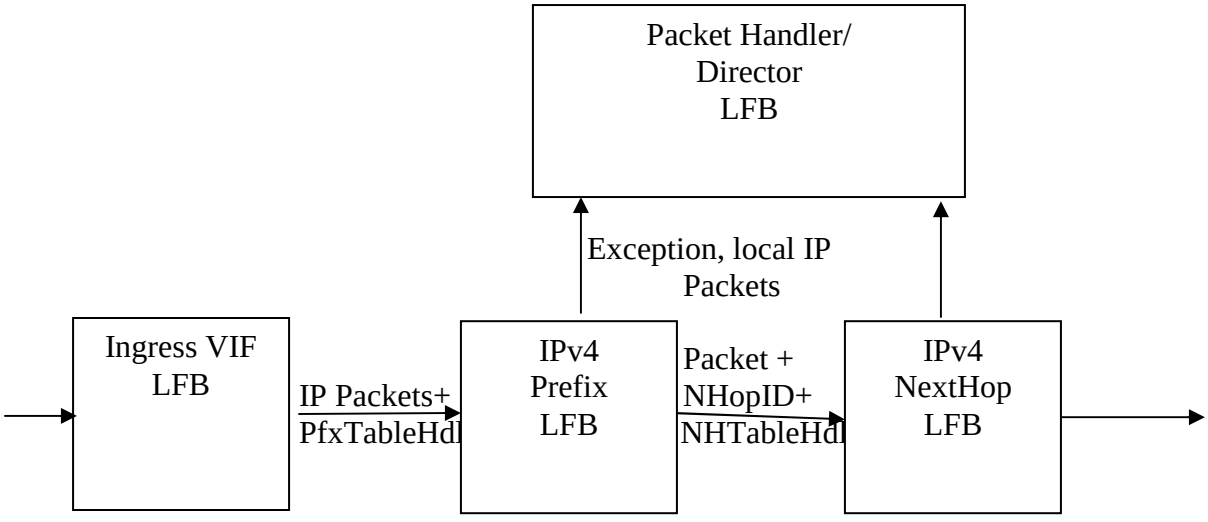

Figure 2-2 IPv4 Prefix LFB interacting with Packet Handler and Next Hop LFB

4 Data Types

4.1 Common LFB Data Types

4.1.1 Prefix LFB Type (as defined in [FAPITOPO])

```
#define NPF_LFB_TYPE_IPv4_PREFIX          10
```

4.1.2 IPv4 Prefix Types

```
#define NPF_LFB_TYPE_IPv4_PREFIX 10
```

This defines the IPv4 Prefix LFB type.

```
typedef struct {
    NPF_IPv4Address_t  IPv4Addr;
    NPF_uint8_t        IPv4NetPlen;
} NPF_IPv4UC_Prefix_t;
```

This is a common structure defined in [IFMSAPI].

```
typedef struct {
    NPF_uint64_t        packetsForwarded;
    NPF_uint64_t        packetsDropped;
    NPF_uint32_t        packetsNoRoute;
} NPF_F_IPv4_PrefixStats_t;
```

This structure defines the statistics associated with the Prefix LFB.

4.1.3 Prefix Table Handle

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

```
typedef NPF_uint32_t NPF_F_IPv4_PrefixTableHandle_t;
```

This table handle is queried from the FAPI implementation.

4.1.4 Metadata

```
#define NPF_META_NEXT_HOP_ID  41
```

This defines the Next Hop metadata tag.

```
typedef enum {
      NPF_F_IPv4_PREFIX_EXCEPTION_RESERVED                = 0,
      NPF_F_IPv4_PREFIX_EXCEPTION_HDR_CHK_FAILED          = 1,
      NPF_F_IPv4_PREFIX_EXCEPTION_RPF_FAILED              = 2
} NPF_F_IPv4_PrefixException_t;
```

This defines the Prefix Exception Id metadata.

4.1.5 Prefix Query Request

```
typedef struct {
   NPF_IPv4Prefix_t                boundingPrefix;
   NPF_IPv4Prefix_t                nextPrefix;
   NPF_F_IPv4PrefixQueryResp_t     *buffer;
   NPF_uint32_t                    bufsize;
} NPF_F_IPv4PrefixQueryRequest_t;
```

The Prefix Query Request structure is used in the NPF_F_IPv4_PrefixQueryEntires function as a means for storing the data returned by this function. The entries returned by this function consist of all prefixes matching the “boundingPrefix” value, beginning at the “nextPrefix” value. The “buffer” member of this structure points to a location in client memory where the data is to be written, and “bufsize” is an indication of the amount of memory that is available to the function for writing the data, as a number of prefix query response structures.

NOTE: If the calling application fails to allocate enough memory to hold all the data returned by the NPF_F_IPv4_PrefixQueryEntires function, the client can request the data not returned by repeating the request with a new “nextPrefix” value, which should be equal to the last prefix returned in the buffer, and repeat this action until all prefixes are returned.

4.1.6 Prefix Query Response

```
typedef struct {
      NPF_IPv4UC_Prefix_t     prefixId;
      NPF_uint32_t            nexthopInfo;
} NPF_F_IPv4PrefixQueryResp_t;
```

This structure defines the prefix query response. It includes the prefix ID and the Next Hop ID.

```
typedef struct {
      NPF_uint32_t                numEntries;
      NPF_boolean_t               moreData;
} NPF_F_IPv4PrefixQueryEntriesResp_t;
```

This structure defines the response for the NPF_F_IPv4PrefixQueryEntries function. It contains the number of entries returned and a boolean value, which if set, indicates that there is more data to be retrieved.

```
typedef struct {
      NPF_uint32_t            maxTableHdls;
      NPF_uint32_t            maxTableSize;
} NPF_F_IPv4_PrefixTableQueryResp_t;
```

This function defines the response for the prefix table query. It consists of a conservative estimate of the maximum number of table handles and the maximum table size that can be created on the FE.

4.2 Data Structures for Completion Callbacks

A completion callback is defined for each of the functions in this API.

4.2.1 Asynchronous Response

This is the asynchronous response structure, which is part of the CallbackData structure that is passed to the caller in the asynchronous response. It contains an error/success code, and a function-specific structure embedded in a union.

```
typedef struct {
      NPF_F_IPv4_PrefixErrorType_t error;
      union {
            NPF_IPv4UC_Prefix_t              prefix;
            NPF_F_IPv4_PrefixTableHandle_t   tbHandle;
            NPF_F_IPv4_PrefixStats_t         prefixStats;
            NPF_uint32_t                     freeEntries;
            NPF_F_IPv4PrefixQueryResp_t      prefixQueryResp;
            NPF_F_NHTableHandle_t            nhopTbHdl;
            NPF_F_IPv4PrefixQueryEntriesResp_t  prefixQueryEntriesResp;
            NPF_F_IPv4_PrefixTableQueryResp_t  maxTblHdls;
      } u;
} NPF_F_IPv4_PrefixAsyncResponse_t;
```

4.2.2 Callback Type

The callback response contains one of the following codes, indicating the function that was called to cause the callback. This code tells the application how to interpret the data included in the union that is part of the response structure.

```
/* completion callback types */
typedef enum NPF_F_IPv4PrefixCallbackType {
      NPF_F_IPv4_PREFIX_TBHANDLE_CREATE   = 1,
      NPF_F_IPv4_PREFIX_TBHANDLE_DELETE   = 2,
      NPF_F_IPv4_PREFIX_ADD               = 3,
      NPF_F_IPv4_PREFIX_DELETE            = 4,
      NPF_F_IPv4_PREFIX_PURGE             = 5,
      NPF_F_IPv4_PREFIX_TBHANDLE_BIND     = 6,
      NPF_F_IPv4_PREFIX_GETSTATS          = 7,
      NPF_F_IPv4_PREFIX_TBATTR_QUERY      = 8,
      NPF_F_IPv4_PREFIX_QUERY             = 9,
      NPF_F_IPv4_PREFIX_QUERY_ENTRIES     = 10,
      NPF_F_IPv4_PREFIX_TABLE_BIND_QUERY  = 11,
      NPF_F_IPv4_PREFIX_TABLE_QUERY       = 12
} NPF_F_IPv4_PrefixCallbackType_t;
```

4.2.3 Callback Data

This is the callback response structure, which is passed to the caller in the asynchronous response from a function call. It contains the callback type that identifies the function called, the allOK flag, the number of responses and an array of response structures depending on the call. This structure is similar to the NPF_IPv4UC_CallbackData_t structure in [IPv4SAPI] and the allOK flag has the same usage in this structure.

```
typedef struct {
    NPF_F_IPv4_PrefixCallbackType_t     type;
    NPF_boolean_t                       allOK;
    NPF_uint32_t                        numResp;
    NPF_F_IPv4_PrefixAsyncResponse_t   *resp;
} NPF_F_IPv4_PrefixCallbackData_t;
```

Table 3-2. Callback type to Callback data mapping table

Callback Type	Callback Data
NPF_F_IPv4_PREFIX_ADD	NPF_IPv4UC_Prefix_t
NPF_F_IPv4_PREFIX_DELETE	NPF_IPv4UC_Prefix_t
NPF_F_IPv4_PREFIX_PURGE	Unused
NPF_F_IPv4_PREFIX_GETSTATS	NPF_F_IPv4_PrefixStats_t
NPF_F_IPv4_PREFIX_TBHANDLE_CREATE	NPF_F_IPv4_PrefixTableHandle_t
NPF_F_IPv4_PREFIX_TBATTR_QUERY	NPF_uint32_t
NPF_F_IPv4_PREFIX_QUERY	NPF_F_IPv4PrefixQueryResp_t
NPF_F_IPv4_PREFIX_QUERY_ENTRIES	NPF_F_IPv4PrefixQueryEntriesResp_t
NPF_F_IPv4_PREFIX_TABLE_BIND_QUERY	NPF_F_NHTableHandle_t
NPF_F_IPv4_PREFIX_TBHANDLE_DELETE	Unused
NPF_F_IPv4_PREFIX_TBHANDLE_BIND	Unused
NPF_F_IPv4_PREFIX_TABLE_QUERY	NPF_F_IPv4_PrefixTableQueryResp_t

4.3 Data Structures for Event Notifications

The following sections detail the information related to the Prefix 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_PREFIX_TABLE_MISS = 1,
   NPF_F_PREFIX_ENTRY_MISS = 2
} NPF_F_IPv4_PrefixEvent_t;
```

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

The entry miss event is triggered when the forwarding plane is unable to find a Prefix Entry within a Prefix Table for a specific packet forwarded. This event is optional.

```
/*
 * Definitions for IPv4 Prefix events to be
 * used in event Mask.
 */
#define NPF_F_PREFIX_TABLE_MISS     (1 << 1)
#define NPF_F_PREFIX_ENTRY_MISS     (1 << 2)
```

4.3.2 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 Prefix LFB. An event type field indicates which member of the union is relevant in the structure.

```
typedef struct {
   NPF_F_IPv4_PrefixEvent_t   type;
    union {
      NPF_F_IPv4_PrefixTableHandle_t      tbHandle;
      NPF_IPv4UC_Prefix_t                 prefix;
    } u;
} NPF_F_IPv4_PrefixEventInfo_t;
```

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

```
typedef struct {
   NPF_uint32_t                    numEvents;
   NPF_F_IPv4_PrefixEventInfo_t    *eventArray;
} NPF_F_IPv4_PrefixEventArray_t;
```

4.4 Error Codes

4.4.1 Common NPF Error Codes

NPF common error codes, defined in [SWAPICON], are listed below:

- **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).

4.4.2 LFB Specific Error Codes

```
      #define NPF_F_IPv4_BASE_ERR 900 /* Base value of 900 wrt other NPF
codes */

      /* Invalid FE Handle */
      #define NPF_E_IPv4_INVALID_FE_HDL   (NPF_F_IPv4_BASE_ERR+1)

      /* Invalid LFB Handle */
      #define NPF_E_IPv4_INVALID_LFB_HDL  (NPF_F_IPv4_BASE_ERR+2)

      /* Invalid Table Handle */
      #define NPF_E_IPv4_INVALID_TB_HDL   (NPF_F_IPv4_BASE_ERR+3)

      /* Invalid Buffer Address */
      #define NPF_E_IPv4_INVALID_BUF_ADDR (NPF_F_Ipv4_BASE_ERR+4)

      typedef NPF_uint32_t    NPF_F_IPv4_PrefixErrorType_t;
```

This defines the asynchronous error codes returned in the function callbacks.

5 Functional APIs (FAPIs)

5.1 Completion Callback APIs

5.1.1 Completion Callback Function

Syntax

```
typedef void (*NPF_F_IPv4_PrefixCallBackFunc_t)(
      NPF_IN NPF_userContext_t            userContext,
      NPF_IN NPF_correlator_t             correlator,
      NPF_IN NPF_F_IPv4_PrefixCallbackData_t    ipv4CallbackData);
```

Description

This callback function is for the application to register the asynchronous response handling routine to the NPF FAPI IPv4 API implementation. This callback function is intended to be implemented by the application, and be registered to the NPF FAPI IPv4 API implementation through NPF_F_IPv4Register( ) 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 IPv4 API function call was made. The correlator is used by the application mainly to distinguish between multiple invocations of the same function.
- ipv4CallbackData
  Response information related to the FAPI IPv4 API function call. Contains information that is common among all functions, as well as information that is specific to a particular function. See NPF_F_IPv4CallbackData_t definition 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_IPv4_PrefixRegister(
      NPF_IN NPF_userContext_t            userContext,
      NPF_IN NPF_F_IPv4_PrefixCallbackFunc_t    ipv4CallbackFunc,
      NPF_OUT NPF_callbackHandle_t  *ipv4CallbackHandle);
```

Description

This function is used by an application to register its completion callback function for receiving asynchronous responses related to NPF FAPI IPv4 API function calls. An application may register multiple callback functions using this function. The callback function is identified by the pair of userContext and ipv4CallbackFunc, and for each individual pair, a unique ipv4CallbackHandle will be assigned for future reference. Since the callback function is identified by both userContext and ipv4CallbackFunc, duplicate registration of the same callback function, each with a different userContext is allowed. Also, the same userContext can be shared among different callback functions. Duplicate registration of the same userContext and ipv4CallbackFunc pair has no effect, and will output a handle that is already assigned to the pair, and will return NPF_E_ALREADY_REGISTERED.

Note : NPF_F_IPv4Register( ) is a synchronous function and has no completion callback associated with it.

Input Parameters

- userContext
  A context item 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. An application can assign any value to the userContext where the value is completely opaque to the NPF FAPI IPv4 API implementation.
- ipv4CallbackFunc
  The pointer to the completion callback function to be registered.

Output Parameters

- ipv4CallbackHandle
  A unique identifier assigned for the registered userContext and ipv4CallbackFunc pair. This handle will be used by the application to specify which callback function to be called when invoking asynchronous NPF FAPI IPv4 API functions. It will also be used when de-registering the userContext and ipv4CallbackFunc pair.

Return Values

- NPF_NO_ERROR
  The registration completed successfully.
- NPF_E_BAD_CALLBACK_FUNCTION

ipv4CallbackFunc is NULL.

- NPF_E_ALREADY_REGISTERED
  No new registration was made since the userContext and ipv4CallbackFunc 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_IPv4_PrefixDeregister(
      NPF_IN NPF_callbackHandle_t   ipv4CallbackHandle);
```

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_IPv4EventRegister( ) is a synchronous function and has no completion callback associated with it.

Input Parameters

- ipv4CallbackHandle
  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.

5.2 Event Notification

This section defines the registration and de-registration functions used to install and remove an event handler routine.

5.2.1 NPF_F_IPv4_PrefixEventCallFunc_t

```
typedef void (*NPF_F_IPv4_PrefixEventCallFunc_t) (
   NPF_IN NPF_userContext_t       userContext,
   NPF_IN NPF_F_IPv4_PrefixEventArray_t data);
```

Description

This function is a registered event notification routine for handling IPv4 Prefix FAPI events.

Input Parameters

- userContext - The context item that was supplied by the application when the event callback routine was registered.
- data – A structure containing an array of event data structures and a count to indicate how many events are present. Each of these NPF_F_prefixEventData_t members contains event specific information and a type field to identify the particular event.

Output Parameters

None.

Return Value

None.

5.2.2 NPF_F_IPv4_PrefixEventRegister

```
NPF_error_t NPF_F_IPv4_PrefixEventRegister(
   NPF_IN NPF_userContext_t           userContext,
   NPF_IN NPF_F_IPv4_PrefixEventCallFunc_t eventCallFunc,
   NPF_IN NPF_eventMask_t            eventMask,
   NPF_OUT NPF_callbackHandle_t     *eventCallHandle);
```

Description

This function is used by an application to register its event handling routine for receiving notifications of IPv4 Prefix events. Applications MAY register multiple event handling routines using this function. The event handling routine is identified by the pair of userContext and eventCallFunc, and for each individual pair, a unique eventCallHandle will be assigned for future reference.

Since the event handling routine is identified by both userContext and eventCallFunc, duplicate registration of the same event handling routine with a different userContext is allowed. Also, the same userContext can be shared among different event handling routines. Duplicate registration of the same userContext and eventCallFunc pair has no effect, will output a handle that is already assigned to the pair and will return NPF_E_ALREADY_REGISTERED.

Input Parameters

- userContext – A context item for uniquely identifying the context of the application registering the event handling routine. The exact value will be provided back to the registered event handling routine as its first parameter when it is called. Applications can assign any value to the userContext and the value is completely opaque to the IPv4 Prefix FAPI implementation
- eventCallFunc – The pointer to the event handling routine to be registered.
- eventMask – This is a bit mask of the IPv4 Prefix events. It allows the application to register for those selected events.

Output Parameters

- eventCallHandle - A unique identifier assigned for the registered userContext and eventCallFunc pair. This handle will be used when deregistering the userContext and eventCallFunc pair.

Return Values

- NPF_NO_ERROR - The registration completed successfully.
- NPF_E_BAD_CALLBACK_FUNCTION – The eventCallFunc is NULL, or otherwise invalid.
- NPF_E_CALLBACK_ALREADY_REGISTERED – No new registration was made since the userContext and eventCallFunc pair was already registered.

5.2.3 NPF_F_IPv4_PrefixEventDeregister

```
NPF_error_t NPF_F_IPv4_PrefixEventDeregister(
   NPF_IN NPF_callbackHandle_t eventCallHandle);
```

Description

This function is used by an application to de-register an event handler routine which was previously registered to receive notifications of IPv4 Prefix events. It represents a unique user context and event handling routine pair.

Input Parameters

- eventCallHandle - The unique identifier returned to the application when the event callback routine was registered.

Output Parameters

None.

Return Values

- NPF_NO_ERROR - The de-registration completed successfully.
- NPF_E_BAD_CALLBACK_HANDLE – The de-registration did not complete successfully due to problems with the callback handle provided.

5.3 IPv4 Prefix FAPIs

5.3.1 NPF_F_IPv4_PrefixTableHandleCreate()

Syntax

```
NPF_error_t       NPF_F_IPv4_PrefixTableHandleCreate (
                  NPF_IN      NPF_callbackHandle_t    cbHandle,
                  NPF_IN      NPF_correlator_t        correlator,
                  NPF_IN      NPF_errorReporting_t    cbDesired,
                  NPF_IN      NPF_FE_Handle_t         feHandle,
                  NPF_IN      NPF_BlockId_t           nodeHandle,
                  NPF_IN      NPF_uint32_t            tblid);
```

Description

This function is used to create a Prefix Table Handle. The handle is associated with the table ID, which is passed by the application of this FAPI into this function call. The table handle is returned by the FAPI implementation.

Input Parameters

- cbHandle: The callback handle returned by NPF_ftopologyRegister() call.
- correlator: A 32-bit value that will be returned in the callback for this function call.
- cbDesired: The desired level of callback verbosity: always, never, or only upon error.
- feHandle: The FE Handle returned by NPF_ftopologyGetFEInfoList_t() call.
- nodeHandle: The Prefix LFB Handle.
- tblid: The Prefix table ID that is associated with the handle.

Output Parameters

None.

Return Codes

- NPF_NO_ERROR: The function call was accepted, and a callback will occur or has already occurred.
- NPF_E_BAD_CALLBACK_HANDLE: The cbHandle parameter is invalid; no callback will occur.

Asynchronous Response

- The NPF_F_IPv4_PrefixCallbackData_t structure will be returned with NPF_F_IPv4_PrefixTableHandle_t set in the NPF_F_IPv4_PrefixAsyncResponse_t structure.
- In case of error, NPF_E_IPv4_INVALID_FE_HDL or NPF_E_IPv4_INVALID_LFB_HDL could be returned.

Notes

None.

5.3.2 NPF_F_IPv4_PrefixTableHandleDelete()

Syntax

```
NPF_error_t       NPF_F_IPv4_PrefixTableHandleDelete (
                  NPF_IN      NPF_callbackHandle_t    cbHandle,
                  NPF_IN      NPF_correlator_t        correlator,
                  NPF_IN      NPF_errorReporting_t    cbDesired,
                  NPF_IN      NPF_FE_Handle_t         feHandle,
                  NPF_IN      NPF_BlockId_t           nodeHandle,
                  NPF_IN      NPF_F_IPv4_PrefixTableHandle_t tbHandle);
```

Description

This function is used to delete the Prefix Table Handle from the Prefix LFB.

Input Parameters

- cbHandle: The callback handle returned by NPF_ftopologyRegister() call.
- correlator: A 32-bit value that will be returned in the callback for this function call.
- cbDesired: The desired level of callback verbosity: always, never, or only upon error.
- feHandle: The FE Handle returned by NPF_ftopologyGetFEInfoList_t() call.
- nodeHandle: The Prefix LFB Handle.
- tbHandle: The Prefix table handle.

Output Parameters

None.

Return Codes

- NPF_NO_ERROR: The function call was accepted, and a callback will occur or has already occurred.
- NPF_E_BAD_CALLBACK_HANDLE: The cbHandle parameter is invalid; no callback will occur.

Asynchronous Response

- The NPF_F_IPv4_PrefixCallbackData_t structure will be returned with NPF_F_IPv4_PrefixErrorType_t set in the NPF_F_IPv4_PrefixAsyncResponse_t structure.
- In case of error, NPF_E_IPv4_INVALID_FE_HDL, NPF_E_IPv4_INVALID_TB_HDL or NPF_E_IPv4_INVALID_LFB_HDL could be returned.

Notes

None.

5.3.3 NPF_F_IPv4_PrefixAdd()

Syntax

```
NPF_error_t       NPF_F_IPv4_PrefixAdd (
                NPF_IN   NPF_callbackHandle_t          cbHandle,
                NPF_IN   NPF_correlator_t              correlator,
                NPF_IN   NPF_errorReporting_t          cbDesired,
                NPF_IN   NPF_FE_Handle_t               feHandle,
                NPF_IN   NPF_BlockId_t                 nodeHandle,
                NPF_IN   NPF_F_IPv4_PrefixTableHandle_t      tbHandle,
                NPF_IN   NPF_uint32_t                  numEntries,
                NPF_IN   NPF_IPv4UC_Prefix_t           *prefixArray,
                NPF_IN   NPF_uint32_t                  *nextHopIDArray);
```

Description

This function is used to add an array of prefix entries to a Prefix table in an IPv4 Prefix LFB. If a prefix entry is already present in the table, it is replaced with this call.

Input Parameters

- cbHandle: The callback handle returned by NPF_ftopologyRegister() call.
- correlator: A 32-bit value that will be returned in the callback for this function call.
- cbDesired: The desired level of callback verbosity: always, never, or only upon error.
- feHandle: The FE Handle returned by NPF_ftopologyGetFEInfoList_t() call.
- nodeHandle: The Prefix LFB Handle.
- tbHandle: The Prefix table handle.
- numEntries: The number of prefix entries to be added.
- prefixArray: The array of prefix entries to be added.
- nextHopIDArray: The array of Next Hop IDs corresponding to the prefix entries.

Output Parameters

None.

Return Codes

- NPF_NO_ERROR: The function call was accepted, and a callback will occur or has already occurred.
- NPF_E_BAD_CALLBACK_HANDLE: The cbHandle parameter is invalid; no callback will occur.

Asynchronous Response

- The NPF_F_IPv4_PrefixCallbackData_t structure will be returned with NPF_IPv4UC_Prefix_t set in the NPF_F_IPv4_PrefixAsyncResponse_t structure.
- In case of error, NPF_E_IPv4_INVALID_FE_HDL, NPF_E_IPv4_INVALID_TB_HDL or NPF_E_IPv4_INVALID_LFB_HDL could be returned.

Notes

None.

5.3.4 NPF_F_IPv4_PrefixDelete()

Syntax

```
NPF_error_t    NPF_F_IPv4_PrefixDelete (
                        NPF_IN      NPF_callbackHandle_t    cbHandle,
                        NPF_IN      NPF_correlator_t        correlator,
                        NPF_IN      NPF_errorReporting_t    cbDesired,
                        NPF_IN      NPF_FE_Handle_t         feHandle,
                        NPF_IN      NPF_BlockId_t           nodeHandle,
                        NPF_IN      NPF_F_IPv4_PrefixTableHandle_t
   tbHandle,
                        NPF_IN      NPF_uint32_t            numEntries,
                        NPF_IN      NPF_IPv4UC_Prefix_t    *prefixArray);
```

Description

This function is used to delete an array of prefix entries from a Prefix table in an IPv4 Prefix LFB.

Input Parameters

- cbHandle: The callback handle returned by NPF_ftopologyRegister() call.
- correlator: A 32-bit value that will be returned in the callback for this function call.
- cbDesired: The desired level of callback verbosity: always, never, or only upon error.
- feHandle: The FE Handle returned by NPF_ftopologyGetFEInfoList_t() call.
- nodeHandle: The Prefix LFB Handle.
- tbHandle: The Prefix table handle.
- numEntries: The number of prefix entries to be deleted.
- prefixArray: The array of prefix entries to be deleted.

Output Parameters

None.

Return Codes

- NPF_NO_ERROR: The function call was accepted, and a callback will occur or has already occurred.
- NPF_E_BAD_CALLBACK_HANDLE: The cbHandle parameter is invalid; no callback will occur.

Asynchronous Response

- The NPF_F_IPv4_PrefixCallbackData_t structure will be returned with NPF_IPv4UC_Prefix_t set in the NPF_F_IPv4_PrefixAsyncResponse_t structure.
- In case of error, NPF_E_IPv4_INVALID_FE_HDL, NPF_E_IPv4_INVALID_TB_HDL or NPF_E_IPv4_INVALID_LFB_HDL could be returned.

Notes

None.

5.3.5 NPF_F_IPv4_PrefixPurge()

Syntax

```
NPF_error_t    NPF_F_IPv4_PrefixPurge (
                        NPF_IN      NPF_callbackHandle_t    cbHandle,
                        NPF_IN      NPF_correlator_t        correlator,
                        NPF_IN      NPF_errorReporting_t    cbDesired,
                        NPF_IN      NPF_FE_Handle_t         feHandle,
                        NPF_IN      NPF_BlockId_t           nodeHandle,
                        NPF_IN      NPF_F_IPv4_PrefixTableHandle_t
   tbHandle);
```

Description

This function is used to purge the prefix table or remove all entries from the prefix table in the IPv4 Prefix LFB. The prefix table handle is still valid after this function call.

Input Parameters

- cbHandle: The callback handle returned by NPF_ftopologyRegister() call.
- correlator: A 32-bit value that will be returned in the callback for this function call.
- cbDesired: The desired level of callback verbosity: always, never, or only upon error.
- feHandle: The FE Handle returned by NPF_ftopologyGetFEInfoList_t() call.
- nodeHandle: The Prefix LFB Block Handle.
- tbHandle: The Prefix table handle.

Output Parameters

None.

Return Codes

- NPF_NO_ERROR: The function call was accepted, and a callback will occur or has already occurred.
- NPF_E_BAD_CALLBACK_HANDLE: The cbHandle parameter is invalid; no callback will occur.

Asynchronous Response

- The NPF_F_IPv4_PrefixCallbackData_t structure will be returned with NPF_F_IPv4_PrefixErrorType_t set in the NPF_F_IPv4_PrefixAsyncResponse_t structure.
- In case of error, NPF_E_IPv4_INVALID_FE_HDL, NPF_E_IPv4_INVALID_TB_HDL or NPF_E_IPv4_INVALID_LFB_HDL could be returned

Notes

None.

5.3.6 NPF_F_IPv4_PrefixTableBind ()

Syntax

```
NPF_error_t    NPF_F_IPv4_PrefixTableBind (
                        NPF_IN      NPF_callbackHandle_t    cbHandle,
                        NPF_IN      NPF_correlator_t        correlator,
                        NPF_IN      NPF_errorReporting_t    cbDesired,
                        NPF_IN      NPF_FE_Handle_t         feHandle,
                        NPF_IN      NPF_BlockId_t           nodeHandle,
                        NPF_IN      NPF_F_IPv4_PrefixTableHandle_t tbHandle,
                        NPF_IN      NPF_F_NHTableHandle_t    nhtblHandle);
```

Description

This function is used to bind or associate a prefix table handle with a next hop table handle. This function controls the Next Hop Table ID metadata value that is output with each packet.

Input Parameters

- cbHandle: The callback handle returned by NPF_ftopologyRegister() call.
- correlator: A 32-bit value that will be returned in the callback for this function call.
- cbDesired: The desired level of callback verbosity: always, never, or only upon error.
- feHandle: The FE Handle returned by NPF_ftopologyGetFEInfoList_t() call.
- nodeHandle: The Prefix LFB Block Handle.
- tbHandle: The Prefix table handle.
- nhtblHandle: The Next Hop table handle.

Output Parameters

None.

Return Codes

- NPF_NO_ERROR: The function call was accepted, and a callback will occur or has already occurred.
- NPF_E_BAD_CALLBACK_HANDLE: The cbHandle parameter is invalid; no callback will occur.

Asynchronous Response

- The NPF_F_IPv4_PrefixCallbackData_t structure will be returned with NPF_F_IPv4_PrefixErrorType_t set in the NPF_F_IPv4_PrefixAsyncResponse_t structure.
- In case of error, NPF_E_IPv4_INVALID_FE_HDL, NPF_E_IPv4_INVALID_TB_HDL or NPF_E_IPv4_INVALID_LFB_HDL could be returned.

Notes

None.

5.4 Optional Functional APIs

5.4.1 NPF_F_IPv4_PrefixGetStats()

Syntax

```
NPF_error_t    NPF_F_IPv4_PrefixGetStats (
                        NPF_IN      NPF_callbackHandle_t    cbHandle,
                        NPF_IN      NPF_correlator_t        correlator,
                        NPF_IN      NPF_errorReporting_t    cbDesired,
                        NPF_IN      NPF_FE_Handle_t         feHandle,
                        NPF_IN      NPF_BlockId_t           nodeHandle,
                        NPF_IN      NPF_F_IPv4_PrefixTableHandle_t
   tbHandle);
```

Description

This is an optional function used to get statistics from the IPv4 Prefix LFB.

Input Parameters

- cbHandle: The callback handle returned by NPF_ftopologyRegister() call.
- correlator: A 32-bit value that will be returned in the callback for this function call.
- cbDesired: The desired level of callback verbosity: always, never, or only upon error.
- feHandle: The FE Handle returned by NPF_ftopologyGetFEInfoList_t() call.
- nodeHandle: The Prefix LFB Block Handle.
- tbHandle: The Prefix table handle.

Output Parameters

None.

Return Codes

- NPF_NO_ERROR: The function call was accepted, and a callback will occur or has already occurred.
- NPF_E_BAD_CALLBACK_HANDLE: The cbHandle parameter is invalid; no callback will occur.

Asynchronous Response

- The NPF_F_IPv4_PrefixCallbackData_t structure will be returned with NPF_F_IPv4_PrefixStats_t set in the NPF_F_IPv4_PrefixAsyncResponse_t structure.
- In case of error, NPF_E_IPv4_INVALID_FE_HDL, NPF_E_IPv4_INVALID_TB_HDL or NPF_E_IPv4_INVALID_LFB_HDL could be returned.

Notes

None.

5.4.2 IPv4 Prefix Table Query

Syntax

```
NPF_F_error_t NPF_F_IPv4_PrefixTableQuery (
               NPF_IN NPF_callbackHandle_t      callbackHandle,
               NPF_IN NPF_correlator_t          correlator,
               NPF_IN NPF_errorReporting_t      errorReporting,
               NPF_IN NPF_FE_Handle_t           feHandle,
               NPF_IN NPF_BlockId_t             lfbHandle);
```

Description

This function provides the conservative estimate for the maximum number of Prefix tables (table handles) which can be created for the particular FE and Prefix LFB, as well as the maximum table size for each of the tables.

Input Parameters

- cbHandle: The callback handle returned by NPF_ftopologyRegister() call.
- correlator: A 32-bit value that will be returned in the callback for this function call.
- cbDesired: The desired level of callback verbosity: always, never, or only upon error.
- feHandle: The FE Handle returned by NPF_ftopologyGetFEInfoList_t() call.
- nodeHandle: The Prefix LFB Block Handle.

Output Parameters

None.

Return Values

- NPF_NO_ERROR: The function call was accepted, and a callback will occur or has already occurred.
- NPF_E_BAD_CALLBACK_HANDLE: The cbHandle parameter is invalid; no callback will occur.

Asynchronous Response

- The NPF_F_IPv4_PrefixCallbackData_t structure will be returned with NPF_F_IPv4_PrefixTableQueryResp_t set in the NPF_F_IPv4_PrefixAsyncResponse_t structure.
- In case of error, NPF_E_IPv4_INVALID_FE_HDL or NPF_E_IPv4_INVALID_LFB_HDL could be returned.

Notes

None.

5.4.3 IPv4 Prefix Table Attribute Query

Syntax

```
NPF_F_error_t NPF_F_IPv4_PrefixTableAttributeQuery (
               NPF_IN NPF_callbackHandle_t      callbackHandle,
               NPF_IN NPF_correlator_t          correlator,
               NPF_IN NPF_errorReporting_t      errorReporting,
               NPF_IN NPF_FE_Handle_t           feHandle,
               NPF_IN NPF_BlockId_t             lfbHandle,
               NPF_IN NPF_F_IPv4_PrefixTableHandle_t  tbHandle);
```

Description

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

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

Input Parameters

- cbHandle: The callback handle returned by NPF_ftopologyRegister() call.
- correlator: A 32-bit value that will be returned in the callback for this function call.
- cbDesired: The desired level of callback verbosity: always, never, or only upon error.
- feHandle: The FE Handle returned by NPF_ftopologyGetFEInfoList_t() call.
- nodeHandle: The Prefix LFB Block Handle.
- tbHandle – The Prefix Table Handle.

Output Parameters

None.

Return Values

- NPF_NO_ERROR: The function call was accepted, and a callback will occur or has already occurred.
- NPF_E_BAD_CALLBACK_HANDLE: The cbHandle parameter is invalid; no callback will occur.

Asynchronous Response

- The NPF_F_IPv4_PrefixCallbackData_t structure will be returned with NPF_uint32_t set in the NPF_F_IPv4_PrefixAsyncResponse_t structure.
- In case of error, NPF_E_IPv4_INVALID_FE_HDL, NPF_E_IPv4_INVALID_TB_HDL or NPF_E_IPv4_INVALID_LFB_HDL could be returned.

Notes

The estimate returned for this function should be a conservative one, i.e. if this is a dynamic quantity, it should be the greatest number of PrefixEntryAdd operations that can be guaranteed to succeed, based on what is known at the time of the request.

5.4.4 IPv4 Prefix Query

Syntax

```
NPF_F_error_t NPF_F_IPv4_PrefixQuery (
                NPF_IN NPF_callbackHandle_t      callbackHandle,
                NPF_IN NPF_correlator_t          correlator,
                NPF_IN NPF_errorReporting_t      errorReporting,
                NPF_IN NPF_FE_Handle_t           feHandle,
                NPF_IN NPF_BlockId_t             lfbHandle,
                NPF_IN NPF_F_IPv4_PrefixTableHandle_t  tbHandle,
                NPF_IN NPF_uint32_t              numEntries,
                NPF_IN NPF_IPv4UC_Prefix_t       *prefixIdArray);
```

Description

This function queries one or more Prefix Entries in the specified Prefix Table. The prefixIdArray points to an array of Prefix Ids of size numEntries.
If the entries exist, the content of the entries are returned in the completion callback, otherwise an error is assigned in the returnCode field of the NPF_F_PrefixAsyncResponse_t structure.

Input Parameters

- cbHandle: The callback handle returned by NPF_ftopologyRegister() call.
- correlator: A 32-bit value that will be returned in the callback for this function call.
- cbDesired: The desired level of callback verbosity: always, never, or only upon error.
- feHandle: The FE Handle returned by NPF_ftopologyGetFEInfoList_t() call.
- nodeHandle: The Prefix LFB Block Handle.
- tbHandle – The Prefix Table Handle.
- numEntries – The number of elements in the prefixIdArray.
- prefixIdArray – Pointer to an array of Prefix Identifiers as determined by the caller.

Output Parameters

None.

Return Values

- NPF_NO_ERROR: The function call was accepted, and a callback will occur or has already occurred.
- NPF_E_BAD_CALLBACK_HANDLE: The cbHandle parameter is invalid; no callback will occur.

Asynchronous Response

- The NPF_F_IPv4_PrefixCallbackData_t structure will be returned with NPF_F_IPv4PrefixQueryResp_t set in the NPF_F_IPv4_PrefixAsyncResponse_t structure.
- In case of error, NPF_E_IPv4_INVALID_FE_HDL, NPF_E_IPv4_INVALID_TB_HDL or NPF_E_IPv4_INVALID_LFB_HDL could be returned.

Notes

None.

5.4.5 IPv4 Prefix Table Binding Query

Syntax

```
NPF_F_error_t NPF_F_IPv4_PrefixTableBindingQuery (
               NPF_IN NPF_callbackHandle_t      callbackHandle,
               NPF_IN NPF_correlator_t          correlator,
               NPF_IN NPF_errorReporting_t      errorReporting,
               NPF_IN NPF_FE_Handle_t           feHandle,
               NPF_IN NPF_BlockId_t             lfbHandle,
               NPF_IN NPF_F_IPv4_PrefixTableHandle_t  tbHandle);
```

Description

This function queries the Next Hop table handle associated with the Prefix table handle.

Input Parameters

- cbHandle: The callback handle returned by NPF_ftopologyRegister() call.
- correlator: A 32-bit value that will be returned in the callback for this function call.
- cbDesired: The desired level of callback verbosity: always, never, or only upon error.
- feHandle: The FE Handle returned by NPF_ftopologyGetFEInfoList_t() call.
- nodeHandle: The Prefix LFB Block Handle.
- tbHandle – The Prefix Table Handle.

Output Parameters

None

Return Values

- NPF_NO_ERROR: The function call was accepted, and a callback will occur or has already occurred.
- NPF_E_BAD_CALLBACK_HANDLE: The cbHandle parameter is invalid; no callback will occur.

Asynchronous Response

- The NPF_F_IPv4_PrefixCallbackData_t structure will be returned with NPF_F_NHTableHandle_t set in the NPF_F_IPv4_PrefixAsyncResponse_t structure.
- In case of error, NPF_E_IPv4_INVALID_FE_HDL, NPF_E_IPv4_INVALID_TB_HDL or NPF_E_IPv4_INVALID_LFB_HDL could be returned.

Notes

None.

5.4.6 IPv4 Prefix Query Entries

Syntax

```
NPF_F_error_t NPF_F_IPv4_PrefixQueryEntries (
               NPF_IN NPF_callbackHandle_t      callbackHandle,
               NPF_IN NPF_correlator_t          correlator,
               NPF_IN NPF_errorReporting_t      errorReporting,
               NPF_IN NPF_FE_Handle_t           feHandle,
               NPF_IN NPF_BlockId_t             lfbHandle,
               NPF_IN NPF_F_IPv4_PrefixTableHandle_t  tbHandle,
               NPF_IN NPF_IPv4PrefixQueryRequest_t    prefixRequest);
```

Description

This function returns all the prefix entries in the specified Prefix Table that match a given prefix (the “bounding prefix” variable in the prefixRequest parameter. The function copies the prefix entities into client memory, as specified by the `prefixRequest` parameter, until it has copied all the matching entries, or it reaches the end of the buffer; which ever comes first.

A call to this function may result in the function having more data to return than the amount of memory allocated by the client. If this is the case, the callback will indicate that there is still data to be retrieved to fully satisfy the request.

On the initial call for a range of entries, the boundingPrefix and nextPrefix variables of the prefixRequest argument should be set to the same value – the value to match. Subsequent calls to this function can be made, incrementally modifying the contents of the `prefixRequest` structure, to retrieve the remaining data. To retrieve the next buffer of prefix entries, the client would replace the “nextPrefix” variable in the prefixRequest argument with the prefix value of the last entry added to the buffer by the previous call. Because this value is now different from the boundingPrefix value, the implementation will begin filling the buffer with the prefix table entry following the nextPrefix entry.

As an example, to retrieve all entries in a prefix table, the client would call NPF_F_IPv4PrefixQueryEntries() with both boundingPrefix and nextPrefix set to 0.0.0.0/0. Every prefix in the table matches this specification, so all prefixes would be returned in successive calls, one buffer-full at a time.

Input Parameters

- cbHandle: The callback handle returned by NPF_ftopologyRegister() call.
- correlator: A 32-bit value that will be returned in the callback for this function call.
- cbDesired: The desired level of callback verbosity: always, never, or only upon error.
- feHandle: The FE Handle returned by NPF_ftopologyGetFEInfoList_t() call.
- nodeHandle: The Prefix LFB Block Handle.
- tbHandle: The Prefix Table Handle.
- prefixRequest: A structure which contains the information required for the callback to write the results of the query to client memory.

Output Parameters

None.

Return Values

- NPF_NO_ERROR: The function call was accepted, and a callback will occur or has already occurred.
- NPF_E_BAD_CALLBACK_HANDLE: The cbHandle parameter is invalid; no callback will occur.

Asynchronous Response

- The NPF_F_IPv4_PrefixCallbackData_t structure will be returned with NPF_F_IPv4PrefixQueryEntriesResp_t set in the NPF_F_IPv4_PrefixAsyncResponse_t structure, containing the number of entries returned and an indication of whether there is more data to be retrieved for the request.
- In case of error, NPF_E_IPv4_INVALID_FE_HDL, NPF_E_IPv4_INVALID_TB_HDL, NPF_E_IPv4_INVALID_LFB_HDL or NPF_E_IPv4_INVALID_BUF_ADDR could be returned.

Notes

None.

6 References

[FORCESREQ]"Requirements for Separation of IP Control and Forwarding", H. Khosravi, T. Anderson et al, RFC 3654.

[FAPITOPO] "Topology Manager Functional API Implementation Agreement Revision 1.0", Network Processing Forum.

[SWAPICON] "Software API Conventions Implementation Agreement Revision 1.0", Network Processing Forum.

[IPv4SAPI] "IPv4 Unicast Forwarding Service API Implementation Agreement Revision 2.0", Network Processing Forum.

[IFMSAPI] "Interface Management API Implementation Agreement Revision 3.0", Network Processing Forum.

[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

APPENDIX A HEADER FILE INFORMATION

```
/*
 * This header file defines typedefs, constants, and functions
 * for the NP Forum Functional IPv4 Prefix API
 */
#ifndef __NPF_F_IPv4_PREFIX_H
#define __NPF_F_IPv4_PREFIX_H

#ifdef __cplusplus
extern "C"      {
#endif

/*---------------------------------------------------------------------
 *
 * Common Data Types
 *
 *---------------------------------------------------------------------*/

typedef struct {
    NPF_IPv4Address_t  IPv4Addr;
    NPF_uint8_t        IPv4NetPlen;
} NPF_IPv4UC_Prefix_t;


typedef struct {
    NPF_uint64_t        packetsForwarded;
    NPF_uint64_t        packetsDropped;
    NPF_uint32_t        packetsNoRoute;
} NPF_F_IPv4_PrefixStats_t;

typedef NPF_uint32_t NPF_F_IPv4_PrefixTableHandle_t;

#define NPF_META_NEXT_HOP_ID  41

typedef enum {
      NPF_F_IPv4_PREFIX_EXCEPTION_RESERVED                  = 0,
      NPF_F_IPv4_PREFIX_EXCEPTION_HDR_CHK_FAILED            = 1,
      NPF_F_IPv4_PREFIX_EXCEPTION_RPF_FAILED                = 2
} NPF_F_IPv4_PrefixException_t;

typedef struct {
      NPF_IPv4Prefix_t              boundingPrefix;
      NPF_IPv4Prefix_t              nextPrefix;
      NPF_F_IPv4PrefixQueryResp_t   *buffer;
      NPF_uint32_t                  bufsize;
} NPF_F_IPv4PrefixQueryRequest_t;

typedef struct {
      NPF_IPv4UC_Prefix_t     prefixId;
      NPF_uint32_t   nexthopInfo;
} NPF_F_IPv4PrefixQueryResp_t;

typedef struct {
      NPF_uint32_t                  numEntries;
      NPF_boolean_t                 moreData;
} NPF_F_IPv4PrefixQueryEntriesResp_t;
```

```
typedef struct {
      NPF_uint32_t            maxTableHdls;
      NPF_uint32_t            maxTableSize;
} NPF_F_IPv4_PrefixTableQueryResp_t;

typedef struct {
      NPF_F_IPv4_PrefixErrorType_t error;
      union {
            NPF_IPv4UC_Prefix_t                 prefix;
            NPF_F_IPv4_PrefixTableHandle_t      tbHandle;
            NPF_F_IPv4_PrefixStats_t            prefixStats;
            NPF_uint32_t                        freeEntries;
            NPF_F_IPv4PrefixQueryResp_t         prefixQueryResp;
            NPF_F_NHTableHandle_t               nhopTbHdl;
            NPF_F_IPv4PrefixQueryEntriesResp_t  prefixQueryEntriesResp;
            NPF_F_IPv4_PrefixTableQueryResp_t   maxTblHdls;
      } u;
} NPF_F_IPv4_PrefixAsyncResponse_t;

/*-------------------------------------------------------------------
 *
 * Completion Callback Data Types
 *
 *-------------------------------------------------------------------*/

/* completion callback types */
typedef enum NPF_F_IPv4PrefixCallbackType {
            NPF_F_IPv4_PREFIX_TBHANDLE_CREATE   = 1,
            NPF_F_IPv4_PREFIX_TBHANDLE_DELETE   = 2,
            NPF_F_IPv4_PREFIX_ADD               = 3,
            NPF_F_IPv4_PREFIX_DELETE            = 4,
            NPF_F_IPv4_PREFIX_PURGE             = 5,
            NPF_F_IPv4_PREFIX_TBHANDLE_BIND     = 6,
            NPF_F_IPv4_PREFIX_GETSTATS          = 7,
            NPF_F_IPv4_PREFIX_TBATTR_QUERY      = 8,
            NPF_F_IPv4_PREFIX_QUERY             = 9,
            NPF_F_IPv4_PREFIX_QUERY_ENTRIES     = 10,
            NPF_F_IPv4_PREFIX_TABLE_BIND_QUERY  = 11,
            NPF_F_IPv4_PREFIX_TABLE_QUERY       = 12
} NPF_F_IPv4_PrefixCallbackType_t;

typedef struct {
    NPF_F_IPv4_PrefixCallbackType_t     type;
    NPF_boolean_t                       allOK;
    NPF_uint32_t                        numResp;
    NPF_F_IPv4_PrefixAsyncResponse_t   *resp;
} NPF_F_IPv4_PrefixCallbackData_t;

/*-------------------------------------------------------------------
 *
 * Event Notification Data Types
 *
 *-------------------------------------------------------------------*/

typedef enum {
   NPF_F_PREFIX_TABLE_MISS = 1,
```

```
   NPF_F_PREFIX_ENTRY_MISS = 2
} NPF_F_IPv4_PrefixEvent_t;

/*
 * Definitions for IPv4 Prefix events to be
 * used in event Mask.
 */
#define NPF_F_PREFIX_TABLE_MISS     (1 << 1)
#define NPF_F_PREFIX_ENTRY_MISS     (1 << 2)

typedef struct {
   NPF_F_IPv4_PrefixEvent_t               type;
    union {
      NPF_F_IPv4_PrefixTableHandle_t      tbHandle;
      NPF_IPv4UC_Prefix_t                 prefix;
    } u;
} NPF_F_IPv4_PrefixEventInfo_t;

typedef struct {
   NPF_uint32_t                     numEvents;
   NPF_F_IPv4_PrefixEventInfo_t     *eventArray;
} NPF_F_IPv4_PrefixEventArray_t;

#define NPF_F_IPv4_BASE_ERR 900 /* Base value of 900 wrt other NPF codes */

#define NPF_E_IPv4_INVALID_FE_HDL   (NPF_F_IPv4_BASE_ERR+1)
#define NPF_E_IPv4_INVALID_LFB_HDL  (NPF_F_IPv4_BASE_ERR+2)
#define NPF_E_IPv4_INVALID_TB_HDL   (NPF_F_IPv4_BASE_ERR+3)
#define NPF_E_IPv4_INVALID_BUF_ADDR (NPF_F_Ipv4_BASE_ERR+4)

typedef NPF_uint32_t    NPF_F_IPv4_PrefixErrorType_t;

/*-------------------------------------------------------------------
 *
 * Function Call Prototypes
 *
 *-------------------------------------------------------------------*/

typedef void (*NPF_F_IPv4_PrefixCallBackFunc_t)(
            NPF_IN NPF_userContext_t            userContext,
            NPF_IN NPF_correlator_t             correlator,
            NPF_IN NPF_F_IPv4_PrefixCallbackData_t    ipv4CallbackData);

NPF_error_t NPF_F_IPv4_PrefixRegister(
            NPF_IN NPF_userContext_t            userContext,
            NPF_IN NPF_F_IPv4_PrefixCallbackFunc_t    ipv4CallbackFunc,
            NPF_OUT NPF_callbackHandle_t  *ipv4CallbackHandle);

NPF_error_t NPF_F_IPv4_PrefixDeregister(
            NPF_IN NPF_ callbackHandle_t  ipv4CallbackHandle);

typedef void (*NPF_F_IPv4_PrefixEventCallFunc_t) (
            NPF_IN NPF_userContext_t       userContext,
            NPF_IN NPF_F_IPv4_PrefixEventArray_t data);

NPF_error_t NPF_F_IPv4_PrefixEventRegister(
            NPF_IN NPF_userContext_t           userContext,
```

```
            NPF_IN NPF_F_IPv4_PrefixEventCallFunc_t eventCallFunc,
            NPF_IN NPF_eventMask_t            eventMask,
            NPF_OUT NPF_callbackHandle_t     *eventCallHandle);

NPF_error_t NPF_F_IPv4_PrefixEventDeregister(
               NPF_IN NPF_callbackHandle_t eventCallHandle);

NPF_error_t    NPF_F_IPv4_PrefixTableHandleCreate (
               NPF_IN   NPF_callbackHandle_t    cbHandle,
               NPF_IN   NPF_correlator_t        correlator,
               NPF_IN   NPF_errorReporting_t    cbDesired,
               NPF_IN   NPF_FE_Handle_t         feHandle,
               NPF_IN   NPF_BlockId_t           nodeHandle,
               NPF_IN   NPF_uint32_t            tblid);

NPF_error_t    NPF_F_IPv4_PrefixTableHandleDelete (
               NPF_IN   NPF_callbackHandle_t    cbHandle,
               NPF_IN   NPF_correlator_t        correlator,
               NPF_IN   NPF_errorReporting_t    cbDesired,
               NPF_IN   NPF_FE_Handle_t         feHandle,
               NPF_IN   NPF_BlockId_t           nodeHandle,
               NPF_IN   NPF_F_IPv4_PrefixTableHandle_t tbHandle);

NPF_error_t    NPF_F_IPv4_PrefixAdd (
               NPF_IN   NPF_callbackHandle_t          cbHandle,
               NPF_IN   NPF_correlator_t              correlator,
               NPF_IN   NPF_errorReporting_t          cbDesired,
               NPF_IN   NPF_FE_Handle_t               feHandle,
               NPF_IN   NPF_BlockId_t                 nodeHandle,
               NPF_IN   NPF_F_IPv4_PrefixTableHandle_t      tbHandle,
               NPF_IN   NPF_uint32_t                  numEntries,
               NPF_IN   NPF_IPv4UC_Prefix_t           *prefixArray,
               NPF_IN   NPF_uint32_t                  *nextHopIDArray);

NPF_error_t    NPF_F_IPv4_PrefixDelete (
               NPF_IN   NPF_callbackHandle_t    cbHandle,
               NPF_IN   NPF_correlator_t        correlator,
               NPF_IN   NPF_errorReporting_t    cbDesired,
               NPF_IN   NPF_FE_Handle_t         feHandle,
               NPF_IN   NPF_BlockId_t           nodeHandle,
               NPF_IN   NPF_F_IPv4_PrefixTableHandle_t      tbHandle,
               NPF_IN   NPF_uint32_t            numEntries,
               NPF_IN   NPF_IPv4UC_Prefix_t    *prefixArray);

NPF_error_t    NPF_F_IPv4_PrefixPurge (
               NPF_IN   NPF_callbackHandle_t    cbHandle,
               NPF_IN   NPF_correlator_t        correlator,
               NPF_IN   NPF_errorReporting_t    cbDesired,
               NPF_IN   NPF_FE_Handle_t         feHandle,
               NPF_IN   NPF_BlockId_t           nodeHandle,
               NPF_IN   NPF_F_IPv4_PrefixTableHandle_t      tbHandle);

NPF_error_t    NPF_F_IPv4_PrefixTableBind (
               NPF_IN   NPF_callbackHandle_t    cbHandle,
               NPF_IN   NPF_correlator_t        correlator,
               NPF_IN   NPF_errorReporting_t    cbDesired,
               NPF_IN   NPF_FE_Handle_t         feHandle,
```

```
                NPF_IN   NPF_BlockId_t           nodeHandle,
                NPF_IN   NPF_F_IPv4_PrefixTableHandle_t tbHandle,
                NPF_IN   NPF_F_NHTableHandle_t    nhtblHandle);

NPF_error_t     NPF_F_IPv4_PrefixGetStats (
                NPF_IN   NPF_callbackHandle_t    cbHandle,
                NPF_IN   NPF_correlator_t        correlator,
                NPF_IN   NPF_errorReporting_t    cbDesired,
                NPF_IN   NPF_FE_Handle_t         feHandle,
                NPF_IN   NPF_BlockId_t           nodeHandle,
                NPF_IN   NPF_F_IPv4_PrefixTableHandle_t      tbHandle);

NPF_F_error_t NPF_F_IPv4_PrefixTableQuery (
                NPF_IN NPF_callbackHandle_t      callbackHandle,
                NPF_IN NPF_correlator_t          correlator,
                NPF_IN NPF_errorReporting_t      errorReporting,
                NPF_IN NPF_FE_Handle_t           feHandle,
                NPF_IN NPF_BlockId_t             lfbHandle);

NPF_F_error_t NPF_F_IPv4_PrefixTableAttributeQuery (
                NPF_IN NPF_callbackHandle_t      callbackHandle,
                NPF_IN NPF_correlator_t          correlator,
                NPF_IN NPF_errorReporting_t      errorReporting,
                NPF_IN NPF_FE_Handle_t           feHandle,
                NPF_IN NPF_BlockId_t             lfbHandle,
                NPF_IN NPF_F_IPv4_PrefixTableHandle_t  tbHandle);

NPF_F_error_t NPF_F_IPv4_PrefixQuery (
                NPF_IN NPF_callbackHandle_t      callbackHandle,
                NPF_IN NPF_correlator_t          correlator,
                NPF_IN NPF_errorReporting_t      errorReporting,
                NPF_IN NPF_FE_Handle_t           feHandle,
                NPF_IN NPF_BlockId_t             lfbHandle,
                NPF_IN NPF_F_IPv4_PrefixTableHandle_t  tbHandle,
                NPF_IN NPF_uint32_t              numEntries,
                NPF_IN NPF_IPv4UC_Prefix_t       *prefixIdArray);


NPF_F_error_t NPF_F_IPv4_PrefixTableBindingQuery (
                NPF_IN NPF_callbackHandle_t      callbackHandle,
                NPF_IN NPF_correlator_t          correlator,
                NPF_IN NPF_errorReporting_t      errorReporting,
                NPF_IN NPF_FE_Handle_t           feHandle,
                NPF_IN NPF_BlockId_t             lfbHandle,
                NPF_IN NPF_F_IPv4_PrefixTableHandle_t  tbHandle);

NPF_F_error_t NPF_F_IPv4_PrefixQueryEntires (
                NPF_IN NPF_callbackHandle_t      callbackHandle,
                NPF_IN NPF_correlator_t          correlator,
                NPF_IN NPF_errorReporting_t      errorReporting,
                NPF_IN NPF_FE_Handle_t           feHandle,
                NPF_IN NPF_BlockId_t             lfbHandle,
                NPF_IN NPF_F_IPv4_PrefixTableHandle_t  tbHandle,
                NPF_IN NPF_IPv4PrefixQueryRequest_t    prefixRequest);


#ifdef __cplusplus
```

```
}
#endif

#endif /* __NPF_F_IPv4_PREFIX H */
```

APPENDIX B 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
Arthur Davis, Ericsson
Ellen Deleganes, Intel
Reda Haddad, Ericsson
Zsolt Harazsti, Modular Networks, Inc.
Hormuzd Khosravi, Intel (Document Editor)
Vinoj Kumar, Agere Systems
David Maxwell, IDT
David Putzolu, Intel
John Renwick, Agere Systems

APPENDIX C LIST OF COMPANIES BELONGING TO NPF DURING APPROVAL PROCESS

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