Interface Management API Implementation Agreement (IPv4 and IPv4 Tunnel Interfaces)

Revision 3.0

Editor: John Renwick, Agere Systems, jrenwick@agere.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.

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

105

Revision	Date	Reason for Changes
3.0	11/22/2004	Extracted IPv4 and IPv4 Tunnel support from the Interface Management API Implementation Agreement, revision 2.0. Made editorial changes for clarification and improvement of the text.

2 Introduction

This document defines the IPv4 and IPv4 tunnel interface types under the NPF Interface Management API.

2.1 Assumptions and External Requirements

1. This API assumes the existence of the Interface Management API Core Function Set, and shares all the same assumptions and external requirements of that API.

2.2 Scope

This document is concerned only with definitions and functions supporting IPv4 and IPv4 tunnel interfaces under NPF Interface Management.

2.3 Dependencies

This API shares the same dependences as the Interface Management API Core Function Set.

The NP Forum IPv4 Unicast Forwarding API Implementation Agreement defines the Forwarding Information Base Handle, **`NPF_IPv4UC_FwdTableHandle_t`.**

2.3.1 Definition of attributes and functions provided by the IM API Core

2.3.1.1 NPF_IfMaxPDU_Size_Set

Using the NPF_IfMaxPDU_Size_Set on an IPv4 interface will set the MTU value. If Path MTU discovery (only tunnel type interfaces supported PMTU) is supported (and enabled) or the link has a dynamic MTU value, maxPDU value will be the maximum value allowed on an interface.

If the MaxPDU is left unspecified (zero), the default value on the particular link type should be chosen (par example 1500bytes on Ethernet).

2.3.1.2 Fwd_Enable/Disable

Enable/disabling forwarding on an IPv4 interface will start/end forwarding of packet out on the interface (egress direction). It is possible to receive local destined traffic as well as originated traffic on a disabled interface.

2.3.2 IPv4 Interface

This interface type represents a Layer 3 interface for IPv4, essentially a binding of IP attributes to a physical or logical port. Its application-settable attributes are:

- IP addresses and prefix lengths
- Receive IP multicast addresses
- FIB handle
- An IPv4 interface can have multiple IPv4 addresses on multiple IPv4 networks. The IPv4 interface attributes structure, **`NPF_IfIPv4_t`**, includes both a "Primary IP address" field and an array of IPv4 addresses and corresponding prefixes. The single address field existed in the first version of this Agreement, and the array of addresses was added later. When the array was added, the single primary address field was left in the structure for compatibility with existing applications. Together, these fields form a single set of IPv4 address/prefix pairs for the interface, but they are managed using different function calls. Arriving packets addressed to any of these addresses are considered locally-addressed packets.

2.3.3 IPv4 link capability

The operation of various L3 applications on an IPv4 interface (e.g. routing protocols) as well as in particular IPv4 address configuration and link management operation depend on the capabilities of the underlying link-layer. For that purpose the IM API operates with the following three different link capabilities of IPv4 interfaces:

- **multicast -** an IPv4 link that supports a native mechanism at the link layer for sending a packet to all (i.e., broadcast) or a subset of all neighbors.
- **point-to-point** – an IPv4 link that connects exactly two IPv4 interfaces.
- **non-broadcast multi-access (NBMA)** – an IPv4 link via which more than two interfaces can attach, but that does not support a native form of multicast or broadcast

The link capability of an IPv4 interface is determined from the underlying link-layer and/or from the type of the IPv4 interface. The capability is deduced by the IM implementation and is required to be specified by the IM API implementation in a read-only interface attribute.

Examples: An IPv4 over Ethernet interface has multicast link capability. A configured IPv4-in-IPv4 tunnel interface have point-to-point link capability.

`Multicast Enabled Interfaces`

In this document the term "multicast enabled interface" denotes an IPv4 interface of either multicast or point-to-point link capability.

The point-to-point case is a degenerated multicast scenario as a node always will be communication with one and only one receiver (the other end) over a point-to-point link.

2.3.4 IP-in-IP Tunnel Interfaces

The NPF Interface Management API defines an IP-in-IP tunnel interface as an interface that is tightly

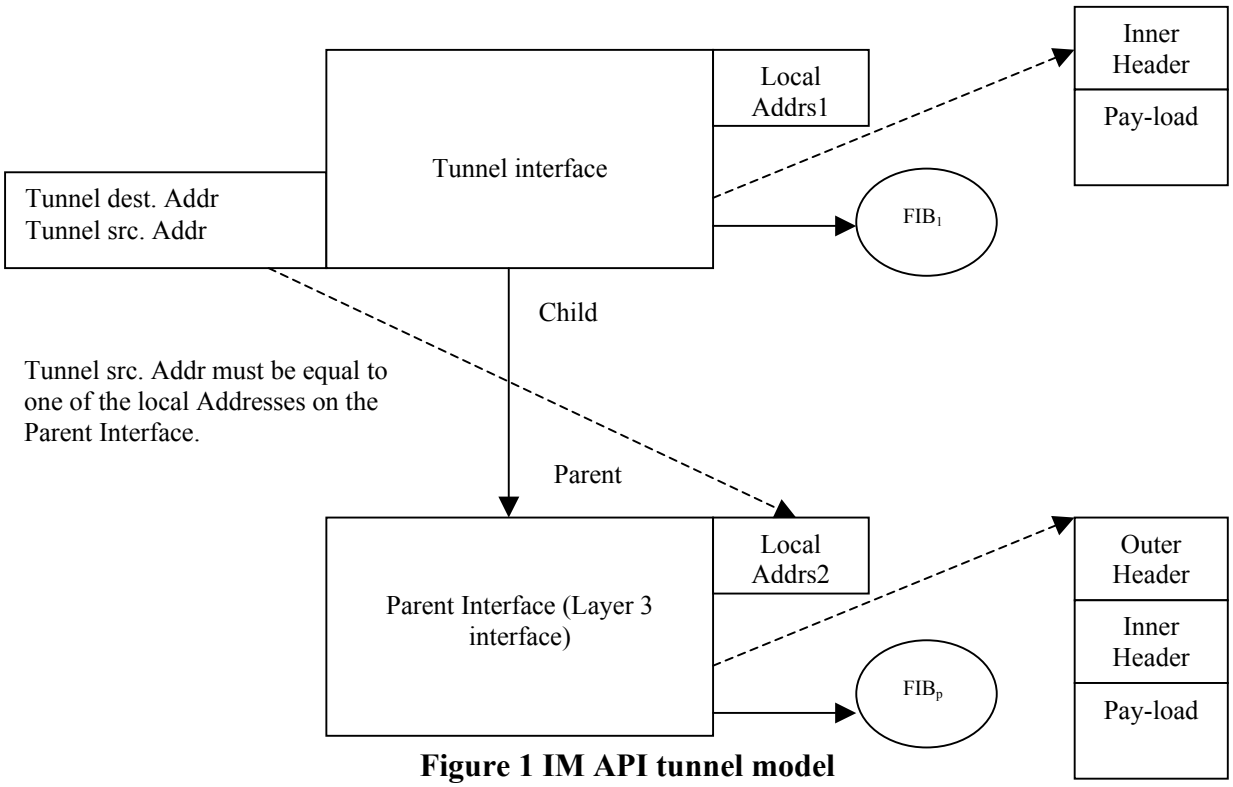

Figure 1 IM API tunnel model

coupled with a parent IP interface (which could be another tunnel, in case of nested tunnels). The tunnel interface has a primary role of encapsulating/decapsulating tunnel packets, while the tightly coupled parent IP interface has the primary role of packet delivery upwards or downwards to the next layer. The encapsulation/decapsulation and packet delivery are performed on the “outer header”, or “tunnel header”. Before encapsulation or after decapsulation the packet is exposing an ‘inner header” and “payload”. This is illustrated in Figure 1.

The relation between the IP-in-IP tunnel interface and its parent IP interface is a normal parent-child relation (with the tunnel as child) with the additional requirement that the tunnel source addresses on which the tunnel interface operates (i.e. the address which is set as IP source address in the encapsulating tunnel header) should be set on the parent IP interface.

An IP interface can have zero or more associated child tunnel interfaces, each identified by the combination of source and destination IP address (on the outer header) and protocol number.

The IM API makes no assumption on the order of interface creation; in particular it is possible to create the tunnel interface without having the parent interface ready. Binding the tunnel interface to the parent interface enables tunnelling on that particular interface.

2.3.4.1 IP-in-IP tunnel interface functionality

The tunnel interface can be instantiated as a full-fledged IP interface with IP addresses of its own (local addr’s in Figure 1). A local IP address of the tunnel interface appears in the inner header of packets originated locally and going out to the tunnel interface.IP-in-IP tunnel tunnel interfaces may also be instantiated as pseudo-interfaces only.

As any standard IP interface an IP-in-IP tunnel interface has a FIB assigned (used for forwarding of decapsulated packets). This FIB may be the same as the one used for the IP parent interface for tunnel interfaces.

2.3.4.2 Incoming Packet Flow

An incoming tunnelled packet arriving at an ordinary IP interface with a destination address equal to one of the nodes interface addresses will be consumed by the node and the appropriate child tunnel interface will be identified based on the outer header addresses and protocol number (as described above).

2.3.4.3 Transmit Packet flow

An outgoing packet arrives on an IP-in-IP tunnel interface as a result of a forwarding decision, i.e. the IP-in-IP tunnel interface is pointed out as the egress interface in the standard manner. After encapsulation the encapsulated packets is handed to the underlying parent IP interface for transmission. This may or may not involve an additional forwarding decision being taken on the encapsulating IP packet using the FIB associated with the parent IPv4 interface.

3 Data Types

3.1 IPv4 Interface Management API Types

3.1.1 Interface Type Code: NPF_IfType_t

The type code (value used in the **`NPF_IfType_t`** variable) for IPv4 interfaces is 5, and for IPv4 tunnel interfaces is 8:

```
#define NPF_IF_TYPE_IPV4            5           /* IPv4 logical interface */
#define NPF_IF_TYPE_TUNNEL_IPV4     6           /* IP-in-IPV4 tunnel */
```

3.1.2 IPv4 link capability: NPF_IfIPv4LinkCapability_t

```
/*
 * IPv4 link capability attribute
 */
typedef enum {
      NPF_IF_IPV4_MULTICAST        = 1,   /* multicast */
      NPF_IF_IPV4_POINT_TO_POINT   = 2,   /* point-to-point */
      NPF_IF_IPV4_NBMA             = 3   /* NBMA */
 }NPF_IfIPv4LinkCapability_t;
```

IP Directed Broadcast Control

This variable type controls Directed Broadcast options for IPv4 interface types. It supports the requirement of section 5.3.5.2 of RFC 1218, as updated by RFC 2644: "A router MAY have an option to enable receiving network-prefix-directed broadcasts on an interface and MAY have an option to enable forwarding network-prefix-directed broadcasts. These options MUST default to blocking receipt and blocking forwarding of network-prefix-directed broadcasts."

```
/*
 *    IP Directed Broadcast Option
 */
typedef enum      {
      NPF_IF_IP_DB_DROP = 1,              /* Drop all directed broadcasts */
      NPF_IF_IP_DB_FORWARD = 2,           /* Enable Forwarding dir. bcasts*/
      NPF_IF_IP_DB_FWD_RCV = 3,           /* Enable forwarding and receive */
      NPF_IF_IP_DB_RECEIVE = 4            /* Enable receipt, no forwarding */
} NPF_IfIpDB_Mode_t;
```

3.1.3 IPv4 Interface Attributes: NPF_IfIPv4_t

The following typedef must be added to npf_if_core.h:

```
typedef struct NPF_IPv4 NPF_IPv4_t;
```

This is the structure definition:

```
/*
```

```
 *    IPv4 Interface attributes
 */
struct NPF_IfIPv4 {
      NPF_IPv4Prefix_t  addr;             /* Primary IPv4 Address and plen */
      NPF_IPv4UC_FwdTableHandle_t  fibHandle;/* Forwarding Info Base*/
      NPF_uint32_t      nUcAddrs;         /* Number of unicast IP addrs */
      NPF_IPv4Prefix_t *ucAddrs;          /* Array of unicast IP addrs */
      NPF_uint32_t      nMcAddrs;         /* Number of mcast IP addrs */
      NPF_IPv4Address_t *mcAddrs;         /* Array of multicast IP addrs */
      NPF_IfIpDB_Mode_t dbMode;           /* Directed Broadcast Control */
      NPF_IfIPv4LinkCapability_t linkCap; /* Link Capability attribute */
};
```

3.1.4 IPv4 Address Prefix: NPF_IPv4Prefix_t

```
/*
 * IPv4 address prefix structure (Defined globally)
 */
typedef struct {
      NPF_IPv4Address_t IPv4Addr;         /* IPv4 address */
      NPF_uint8_t       IPv4Plen;         /* Prefix length in bits (1-32) */
} NPF_IPv4Prefix_t;
```

3.1.5 Tunnel Interface Path MTU config: NPF_IfIPv4_TunnelMTU_t

```
/*
 * Tunnel interface MTU mode
 */
 typedef enum {
      NPF_IFIPv4_TUNNEL_PMTU_DISABLED = 1,
      NPF_IFIPv4_TUNNEL_PMTU_ENABLED = 2
 } NPF_IfIPv4TunnelMTU_t;
```

3.1.6 Using DSCP from inner header: NPF_IfIPv4_InnerDSCP_t

```
/*
 * Tunnel DSCP mode
 */
 typedef enum {
      NPF_IFIPv4_INNER_DSCP_DISABLED = 1,
      NPF_IFIPv4_INNER_DSCP_ENABLED  = 2
 } NPF_IfIPv4InnerDSCP_t;
```

3.1.7 IPv4-in-IPv4 Tunnel Interface Attributes: NPF_IfTunnelIPv4_t

The following typedef must be added to npf_if_core.h:

```
typedef struct NPF_IfTunnelIPv4 NPF_IfTunnelIPv4_t;
```

This is the structure definition:

```
/*
 *    IP-in-IPv4 Tunnel Interface Attributes
 */
struct NPF_IfTunnelIPv4 {
      NPF_IPv4Prefix_t  primAddr;         /* Primary IPv4 address prefix */
```

```
        NPF_uint8_t       maxHops;          /* Maximum hops in the tunnel */
        NPF_IPv4Address_t dstAddr;          /* Tunnel destination IP addr */
        NPF_IPv4Address_t srcAddr;          /* Tunnel source IP addr */
        NPF_uint8_t       DSCP;             /* DiffServ Code Point for hdr */
        NPF_IPv4UC_FwdTableHandle_t fibHandle;/*FIB for forwarding inner hdr*/
        NPF_IfIPv4InnerDSCP_t       IPv4DSCP;/*Use DSCP from inner pck*/
        NPF_IfIPv4TunnelMTU_t       MTU_MODE; /* PMTU on/off */
        NPF_uint32_t                nAddr;  /*Number of IPv4 addresses*/
        NPF_IPv4Prefix_t            *Addrs; /*Array of IPv4 addresses*/
        NPF_uint32_t      nMcAddrs;         /* Number of mcast IP addrs */
        NPF_IPv4Address_t *mcAddrs;         /* Array of multicast IP addrs */
        NPF_IfIPv4LinkCapability_t  linkCap;/* Link Capability attribute */
};
```

3.1.8 Tunnel Address Structure: NPF_IfTunnelIPv4Addr_t

```
/*
 * IPv4 Tunnel Addresses
 */
typedef struct    {
      NPF_IPv4Address_t destAddr;         /* Tunnel destination address */
      NPF_IPv4Address_t srcAddr;          /* Tunnel source address*/
} NPF_IfTunnelIPv4Addr_t;
```

3.2 Data Structures for Completion Callbacks

3.2.1 Completion Callback Type Codes: NPF_IfCallbackType_t

The following codes are used in the **`NPF_IfCallbackType_t`** variable in asynchronous callbacks; this value indicates what function is generating the callback.

```
/*
 *    Completion Callback Types
 */
#define NPF_IF_IPV4ADDR_SET         ((NPF_IF_TYPE_IPV4<<16)+1)
#define NPF_IF_IPV4ADDR_CLEAR       ((NPF_IF_TYPE_IPV4<<16)+2)
#define NPF_IF_IPV4FIB_SET          ((NPF_IF_TYPE_IPV4<<16)+3)
#define NPF_IF_IPV4_UC_ADDR_SET     ((NPF_IF_TYPE_IPV4<<16)+4)
#define NPF_IF_IPV4_UC_ADDR_ADD     ((NPF_IF_TYPE_IPV4<<16)+5)
#define NPF_IF_IPV4_UC_ADDR_DELETE  ((NPF_IF_TYPE_IPV4<<16)+6)
#define NPF_IF_IPV4_MCAST_ADDR_SET  ((NPF_IF_TYPE_IPV4<<16)+7)
#define NPF_IF_IPV4_MCAST_ADDR_ADD  ((NPF_IF_TYPE_IPV4<<16)+8)
#define NPF_IF_IPV4_MCAST_ADDR_DELETE     ((NPF_IF_TYPE_IPV4<<16)+9)
#define NPF_IF_TUNNEL_HOPS_SET      ((NPF_IF_TYPE_IPV4<<16)+10)
#define NPF_IF_TUNNEL_DSCP_SET      ((NPF_IF_TYPE_IPV4<<16)+11)
#define NPF_IF_TUNNEL_IPV4_ADDR_SET ((NPF_IF_TYPE_IPV4<<16)+12)
#define NPF_IF_IPV4_RUNTIME_MTU_GET ((NPF_IF_TYPE_IPV4<<16)+13)
```

3.2.2 Asynchronous Response Array Element: NPF_IfAsyncResponse_t

The **`NPF_IfAsyncResponse_t`** type is defined in the Core Interface Management IA. This structure contains a union. In this union are pointers to various structures returned by Interface Management API functions. If the IPv4 or IPv4 tunnel interface type is supported, the following must be included in the union within the **`NPF_IfAsyncResponse_t`** structure:

```
        NPF_IPv4Prefix_t      *v4prefix;  /* NPF_IPv4UC_AddrAdd(), Set(), */
```

```
                                        /* Delete() */
        NPF_IPv4Address_t     *v4addr;    /* NPF_IPv4McastAddrAdd(), Set() */

        NPF_uint16_t          *MTU_Value  /* NPF_IfIPv4RuntimeMTU_Get */
```

The following table summarizes the information returned by each function in this API.

Function Name	Type Code	Structure Returned
NPF_IfIPv4AddrSet	NPF_IF_IPV4ADDR_SET	Unused
NPF_IfIPv4AddrClear	NPF_IF_IPV4ADDR_CLEAR	Unused
NPF_IfIPv4FIBSet	NPF_IF_IPV4FIB_SET	Unused
NPF_IPv4UC_AddrSet	NPF_IF_IPV4_UC_ADDR_SET	NPF_IPv4Prefix_t *
NPF_IPv4UC_AddrAdd	NPF_IF_IPV4_UC_ADDR_ADD	NPF_IPv4Prefix_t *
NPF_IPv4UC_AddrDelete	NPF_IF_IPV4_UC_ADDR_DELETE	NPF_IPv4Prefix_t *
NPF_IPv4McastAddrSet	NPF_IF_IPV4_MCAST_ADDR_SET	NPF_IPv4Addr_t *
NPF_IPv4McastAddrAdd	NPF_IF_IPV4_MCAST_ADDR_ADD	NPF_IPv4Addr_t *
NPF_IPv4McastAddrDelete	NPF_IF_IPV4_MCAST_ADDR_DELETE	NPF_IPv4Addr_t *
NPF_IPv4TunnelHopsSet	NPF_IF_TUNNEL_HOPS_SET	Unused
NPF_IPv4TunnelDSCP_Set	NPF_IF_TUNNEL_DSCP_SET	Unused
NPF_IfTunnelIPv4AddrSet	NPF_IF_TUNNEL_IPV4_ADDR_SET	Unused
NPF_IfIPv4RuntimeMTU_Get	NPF_IF_IPV4_RUNTIME_MTU_GET	NPF_uint16_t *

3.3 Error Codes

These codes are used as values of the **`NPF_IfErrorType_t`** variable.

```
/*
 *    Asynchronous error codes (returned in function callbacks)
 */

/* Invalid IP address */
#define NPF_IF_IPV4_E_INVALID_IPADDR      ((NPF_IF_TYPE_IPV4<<16) + 1)

/* Invalid IP  prefix length */
#define NPF_IF_IPV4_E_INVALID_PLEN  ((NPF_IF_TYPE_IPV4<<16) + 2)

/* Invalid FIB handle */
#define NPF_IF_IPV4_E_INVALID_FIB_HANDLE  ((NPF_IF_TYPE_IPV4<<16) + 4)

/* Invalid TTL value */
#define NPF_IF_IPV4_E_INVALID_TTL         ((NPF_IF_TYPE_IPV4<<16) + 7)

/* Invalid DSCP value */
#define NPF_IF_IPV4_E_INVALID_DSCP  ((NPF_IF_TYPE_IPV4<<16) + 8)

/* Invalid IPv4 Multicast Address */
#define NPF_IF_IPV4_E_INVALID_IPV4_MCAST_ADDR ((NPF_IF_TYPE_IPV4<<16) + 9)
```

3.4 Data Structures for Event Notifications

3.4.1 Event Types: NPF_IfEvent_t

These codes are used as values of the **`NPF_IfEvent_t`** variable.

```
#define NPF_IF_IPV4_MTU_CHANGE ((NPF_IF_TYPE_IPV4<<16) + 1)
```

The **NPF_IF_IPV4_MTU_CHANGE** event is generated whenever the runtime MTU on the interface changes.

```
#define NPF_IF_IPV4_FIB_CHANGE      ((NPF_IF_TYPE_IPV4<<16) + 2)
```

The **NPF_IF_IPV4_FIB_CHANGE** event occurs whenever a new FIB is assigned to and IPv4 interface.

3.4.2 Event Notification Structure: NPF_IfEventData_t

The following must appear inside the union within the **NPF_IfEventData_t** structure:

```
NPF_uint16_t      *rt_mtu;    /*New runtime MTU value */
NPF_IPv4_FwdTableHandle_t *fibHandle;     /* New FIB handle */
```

4 Functions

4.1 Completion Callback

No completion callback function is defined in this document. All interface type-specific functions use the callback definitions of the Core Interface Management API.

4.2 Event Notification

No event-related functions are defined in this document. All interface type-specific functions use the event function definitions of the Core Interface Management API.

4.3 IPv4 Interface Management API

This section will define functions for querying and modifying the interface properties and attributes.

Note: These functions follow a convention permitting multiple interface handles to be passed for action in a single function invocation. In each case there is an argument that indicates the size of the array of interface handles or addresses. No limit on the size of such arrays is specified by this agreement; however an implementation MAY impose a size limit of its own choosing. If an application exceeds such limit, the implementation SHALL return the response code **`NPF_IF_IPV4_E_BAD_ARRAY_LENGTH`** synchronously.

4.3.1 NPF_IfIPv4AddrSet: Set an IPv4 Interface's IP Address

Syntax

```
NPF_error_t  NPF_IfIPv4AddrSet(
      NPF_IN NPF_callbackHandle_t   if_cbHandle,
      NPF_IN NPF_correlator_t       if_cbCorrelator,
      NPF_IN NPF_errorReporting_t   if_errorReporting,
      NPF_IN NPF_uint32_t           n_handles,
      NPF_IN NPF_IfHandle_t        *if_HandleArray,
      NPF_IN NPF_IPv4Prefix_t      *if_IPv4AddrArray);
```

Description

This function sets the Primary IP address and prefix length of one or more IPv4 interfaces. The **`if_Handle`** and **`if_IPv4Addr`** arrays must both contain the same number of entries, equal to the value of n_handles. The address of each interface is set from a *different* element of the address array.

Input Parameters

- **`if_cbHandle`**: the registered callback handle.
- **`if_cbCorrelator`**: the application's context for this call.
- **`if_errorReporting`**: the desired callback.
- **`n_handles`**: the number of interfaces to set the IP address for.
- **`if_HandleArray`**: pointer to an array of interface handles.
- **`if_IPv4AddrArray`**: pointer to an array of IPv4 address/length structures.

Output Parameters

None

Asynchronous Error Codes

- **NPF_NO_ERROR**: Operation successful.
- **NPF_IF_IPV4_E_INVALID_HANDLE**: An **if_Handle** is null or invalid, or is not an IPv4 interface.
- **NPF_IF_IPV4_E_INVALID_IPADDR**: IP address is not a valid unicast address.
- **NPF_IF_IPV4_E_INVALID_PLEN**: Invalid prefix length.

Asynchronous Response

A total of **n_handles** asynchronous responses (**NPF_IfAsyncResponse_t**) will be passed to the callback function, in one or more invocations. Each response contains an interface handle and a success code or a possible error code for that interface. The union in the callback response structure is unused.

4.3.2 NPF_IfIPv4AddrClear: Clear an IPv4 Interface's Primary IP Address

Syntax

```
NPF_error_t  NPF_IfIPv4AddrClear(
      NPF_IN NPF_callbackHandle_t   if_cbHandle,
      NPF_IN NPF_correlator_t       if_cbCorrelator,
      NPF_IN NPF_errorReporting_t   if_errorReporting,
      NPF_IN NPF_uint32_t           n_handles,
      NPF_IN NPF_IfHandle_t        *if_HandleArray);
```

Description

This function clears the Primary IP address and prefix length of one or more IPv4 interfaces. If the secondary list of IP addresses is empty, the interface has no IP address (it is "unnumbered") after this call is complete. Note that only point-to-point interfaces can function without an address.

Input Parameters

- **if_cbHandle**: the registered callback handle.
- **if_cbCorrelator**: the application's context for this call.
- **if_errorReporting**: the desired callback.
- **n_handles**: the number of interfaces to set the IP address for.
- **if_HandleArray**: pointer to an array of interface handles.

Output Parameters

None

Asynchronous Error Codes

- **NPF_NO_ERROR**: Operation successful.
- **NPF_IF_IPV4_E_INVALID_HANDLE**: An **if_Handle** is null or invalid, or is not an IPv4 interface.

Asynchronous Response

A total of **n_handles** asynchronous responses (**NPF_IfAsyncResponse_t**) will be passed to the callback function, in one or more invocations. Each response contains an interface handle and a success code or a possible error code for that interface. The union in the callback response structure is unused.

4.3.3 NPF_IfIPv4FIB_Set: Associate an IPv4 FIB to an Interface

Syntax

```
NPF_error_t  NPF_IfIPv4FIB_Set(
      NPF_IN NPF_callbackHandle_t              if_cbHandle,
      NPF_IN NPF_correlator_t                  if_cbCorrelator,
      NPF_IN NPF_errorReporting_t              if_errorReporting,
      NPF_IN NPF_uint32_t                      n_handles,
      NPF_IN NPF_IfHandle_t                   *if_HandleArray,
      NPF_IN NPF_IPv4UC_FwdTableHandle_t       if_FIB_Handle);
```

Description

This function associates an IPv4 Forwarding Table (FIB) with one or more IPv4 or IPv4 tunnel interfaces. The new FIB handle becomes the **`if_FIB_Handle`** attribute of the interface. A single FIB is associated with all the interfaces in the **`if_HandleArray`** array.

Input Parameters

- **`if_cbHandle`**: the registered callback handle.
- **`if_cbCorrelator`**: the application's context for this call.
- **`if_errorReporting`**: the desired callback.
- **`n_handles`**: the number of interfaces to set the FIB for.
- **`if_HandleArray`**: pointer to an array of interface handles.
- **`if_FIB_Handle`**: the new FIB handle.

Output Parameters

None

Asynchronous Error Codes

- **`NPF_NO_ERROR`**: Operation complete.
- **`NPF_IF_IPV4_E_INVALID_HANDLE`**: An **`if_Handle`** is null or invalid, or is not an IPv4 interface.
- **`NPF_IF_IPV4_E_INVALID_FIB_HANDLE`**: Invalid FIB handle.

Asynchronous Response

A total of **`n_handles`** asynchronous responses (**`NPF_IfAsyncResponse_t`**) will be passed to the callback function, in one or more invocations. Each response contains an interface handle and a success code or a possible error code for that interface. The union in the callback response structure is unused.

4.3.4 NPF_IfIPv4UC_AddrSet: Set an IPv4 Interface's Secondary List of IP Prefixes

Syntax

```
NPF_error_t  NPF_IfIPv4UC_AddrSet(
      NPF_IN NPF_callbackHandle_t   if_cbHandle,
      NPF_IN NPF_correlator_t       if_cbCorrelator,
      NPF_IN NPF_errorReporting_t   if_errorReporting,
      NPF_IN NPF_IfHandle_t         if_Handle,
      NPF_IN NPF_uint32_t           nAddrs,
      NPF_IN NPF_IPv4Prefix_t       *if_IPv4AddrArray);
```

Description

This function sets (or replaces) an interface's secondary list of unicast IPv4 prefixes. The effect of placing a prefix in this array is to cause arriving packets addressed to that prefix to be delivered locally. If the given array is empty, all secondary uincast IPv4 prefixes are removed.

Input Parameters

- **`if_cbHandle`**: the registered callback handle.
- **`if_cbCorrelator`**: the application's context for this call.
- **`if_errorReporting`**: the desired callback.
- **`if_Handle`**: the handle of the affected interface.
- **`nAddrs`**: the number of entries in the IP prefix array. Set this to zero to remove all prefixes.
- **`if_IPv4AddrArray`**: pointer to an array of IPv4 prefix structures. This may be NULL if **`nAddrs`** is zero.

Output Parameters

None

Asynchronous Error Codes

- **`NPF_NO_ERROR`**: Operation successful.
- **`NPF_IF_IPV4_E_INVALID_HANDLE`**: **`if_Handle`** is null or invalid, or is not an IPv4 interface.
- **`NPF_IF_IPV4_E_INVALID_IPADDR`**: IP address is not a valid unicast address.
- **`NPF_IF_IPV4_E_INVALID_PLEN`**: Invalid prefix length.

Asynchronous Response

If **`nAddrs`** is nonzero, a total of **`nAddrs`** asynchronous responses (**`NPF_IfAsyncResponse_t`**) will be passed to the callback function, in one or more invocations. Each response contains one of the given prefixes and a success code or a possible error code for that prefix. If **`nAddrs`** is given as zero, a single response (**`NPF_IfAsyncResponse_t`**) is returned, containing only a success/failure code.

4.3.5 NPF_IfIPv4UC_AddrAdd: Add to an IPv4 Interface's Secondary List of IP Prefixes

Syntax

```
NPF_error_t  NPF_IfIPv4UC_AddrAdd(
      NPF_IN NPF_callbackHandle_t   if_cbHandle,
      NPF_IN NPF_correlator_t       if_cbCorrelator,
      NPF_IN NPF_errorReporting_t   if_errorReporting,
      NPF_IN NPF_IfHandle_t         if_Handle,
      NPF_IN NPF_uint32_t           nAddrs,
      NPF_IN NPF_IPv4Prefix_t      *if_IPv4AddrArray);
```

Description

This function adds more IPv4 prefixes to an interface's secondary list of unicast IPv4 prefixes. The effect of adding an prefix to this array is to cause arriving packets addressed to that prefix to be delivered locally. If a given prefix is already in the array, or is already the Primary IP address, that prefix is not added a second time.

Input Parameters

- **`if_cbHandle`**: the registered callback handle.
- **`if_cbCorrelator`**: the application's context for this call.

- **if_errorReporting**: the desired callback.
- **if_Handle**: the handle of the affected interface.
- **nAddrs**: the number of entries in the IP prefix array. This must not be zero.
- **if_IPv4AddrArray**: pointer to an array of IPv4 prefix structures.

Output Parameters

None

Asynchronous Error Codes

- **NPF_NO_ERROR**: Operation successful.
- **NPF_IF_IPV4_E_INVALID_HANDLE**: **if_Handle** is null or invalid, or is not an IPv4 interface.
- **NPF_IF_IPV4_E_INVALID_IPADDR**: IP address is not a valid unicast address.
- **NPF_IF_IPV4_E_INVALID_PLEN**: Invalid prefix length.

Asynchronous Response

A total of **nAddrs** asynchronous responses (**NPF_IfAsyncResponse_t**) will be passed to the callback function, in one or more invocations. Each response contains one of the given prefixes and a success code or a possible error code for that prefix.

4.3.6 NPF_IfIPv4UC_AddrDelete: Delete Prefixes From an IPv4 Interface's Secondary List of IP Prefixes

Syntax

```
NPF_error_t  NPF_IfIPv4UC_AddrDelete(
      NPF_IN NPF_callbackHandle_t   if_cbHandle,
      NPF_IN NPF_correlator_t       if_cbCorrelator,
      NPF_IN NPF_errorReporting_t   if_errorReporting,
      NPF_IN NPF_IfHandle_t         if_Handle,
      NPF_IN NPF_uint32_t           nAddrs,
      NPF_IN NPF_IPv4Prefix_t     *if_IPv4AddrArray);
```

Description

This function deletes IPv4 prefixes from an interface's secondary list of unicast IPv4 prefixes.

Input Parameters

- **if_cbHandle**: the registered callback handle.
- **if_cbCorrelator**: the application's context for this call.
- **if_errorReporting**: the desired callback.
- **if_Handle**: the handle of the affected interface.
- **nAddrs**: the number of entries in the IP prefix array. This must not be zero.
- **if_IPv4AddrArray**: pointer to an array of IPv4 prefix structures.

Output Parameters

None

Asynchronous Error Codes

- **NPF_NO_ERROR**: Operation successful.
- **NPF_IF_IPV4_E_INVALID_HANDLE**: **if_Handle** is null or invalid, or is not an IPv4 interface.
- **NPF_IF_IPV4_E_INVALID_IPADDR**: IP address is not a valid unicast address.
- **NPF_IF_IPV4_E_INVALID_PLEN**: Invalid prefix length.

- **NPF_IF_IPV4_E_NO_SUCH_ADDRESS**: Address not found on this interface.

Asynchronous Response

A total of **nAddrs** asynchronous responses (**NPF_IfAsyncResponse_t**) will be passed to the callback function, in one or more invocations. Each response contains one of the given prefixes and a success code or a possible error code for that prefix.

4.3.7 NPF_IfIPv4McastAddrSet: Set an IPv4 Interface's List of Receive Multicast IP Addresses

Syntax

```
NPF_error_t  NPF_IfIPv4McastAddrSet(
      NPF_IN NPF_callbackHandle_t   if_cbHandle,
      NPF_IN NPF_correlator_t       if_cbCorrelator,
      NPF_IN NPF_errorReporting_t   if_errorReporting,
      NPF_IN NPF_IfHandle_t         if_Handle,
      NPF_IN NPF_uint32_t           nAddrs,
      NPF_IN NPF_IPv4Address_t      *mcAddrArray);
```

Description

This function sets (or replaces) an interface's list of receive multicast IPv4 addresses. The effect of placing an address in this array is to cause arriving multicast packets addressed to that address to be delivered locally. If the given array length is zero, all multicast IPv4 addresses are removed.

Input Parameters

- **if_cbHandle**: the registered callback handle.
- **if_cbCorrelator**: the application's context for this call.
- **if_errorReporting**: the desired callback.
- **if_Handle**: the handle of the affected interface.
- **nAddrs**: the number of entries in the IP address array; zero to remove all addresses.
- **mcAddrArray**: pointer to an array of IPv4 multicast addresses. This may be NULL if **nAddrs** is zero.

Output Parameters

None

Asynchronous Error Codes

- **NPF_NO_ERROR**: Operation successful.
- **NPF_IF_IPV4_E_INVALID_HANDLE**: **if_Handle** is null or invalid, or is not an IPv4 interface.
- **NPF_IF_IPV4_E_INVALID_IPMCAST_ADDR**: IP address is not a valid multicast address.

Asynchronous Response

If **nAddrs** is nonzero, a total of **nAddresses** asynchronous responses (**NPF_IfAsyncResponse_t**) will be passed to the callback function, in one or more invocations. Each response contains one of the given addresses and a success code or a possible error code for that address. If **nAddrs** is given as zero, a single response (**NPF_IfAsyncResponse_t**) is returned, containing only a success/failure code.

4.3.8 NPF_IfIPv4McastAddrAdd: Add to an IPv4 Interface's List of Receive Multicast IP Addresses

Syntax

```
NPF_error_t  NPF_IfIPv4McastAddrAdd(
      NPF_IN NPF_callbackHandle_t   if_cbHandle,
      NPF_IN NPF_correlator_t       if_cbCorrelator,
      NPF_IN NPF_errorReporting_t   if_errorReporting,
      NPF_IN NPF_IfHandle_t         if_Handle,
      NPF_IN NPF_uint32_t           nAddrs,
      NPF_IN NPF_IPv4Address_t      *mcAddrArray);
```

Description

This function adds new addresses to an interface's list of receive multicast IPv4 addresses. The effect of adding an address to this array is to cause arriving multicast packets addressed to that address to be delivered locally. If a given address is already in the array, that address is not added a second time.

Input Parameters

- **`if_cbHandle`**: the registered callback handle.
- **`if_cbCorrelator`**: the application's context for this call.
- **`if_errorReporting`**: the desired callback.
- **`if_Handle`**: the handle of the affected interface.
- **`nAddrs`**: the number of entries in the IP address array.
- **`mcAddrArray`**: pointer to an array of IPv4 multicast addresses.

Output Parameters

None

Asynchronous Error Codes

- **`NPF_NO_ERROR`**: Operation successful.
- **`NPF_IF_IPV4_E_INVALID_HANDLE`**: **`if_Handle`** is null or invalid, or is not an IPv4 interface.
- **`NPF_IF_IPV4_E_INVALID_IPMCAST_ADDR`**: IP address is not a valid multicast address.

Asynchronous Response

A total of **`nAddrs`** asynchronous responses (**`NPF_IfAsyncResponse_t`**) will be passed to the callback function, in one or more invocations. Each response contains one of the given addresses and a success code or a possible error code for that address.

4.3.9 NPF_IfIPv4McastAddrDelete: Delete Receive Multicast IPv4 Addresses

Syntax

```
NPF_error_t  NPF_IfIPv4McastAddrDelete(
      NPF_IN NPF_callbackHandle_t   if_cbHandle,
      NPF_IN NPF_correlator_t       if_cbCorrelator,
      NPF_IN NPF_errorReporting_t   if_errorReporting,
      NPF_IN NPF_IfHandle_t         if_Handle,
      NPF_IN NPF_uint32_t           nAddrs,
      NPF_IN NPF_IPv4Address_t      *mcAddrArray);
```

Description

This function removes addresses from an interface's list of receive multicast IPv4 addresses. If a given address is not in the array, no action is taken and no error code is returned.

Input Parameters

- **`if_cbHandle`**: the registered callback handle.
- **`if_cbCorrelator`**: the application's context for this call.
- **`if_errorReporting`**: the desired callback.
- **`if_Handle`**: the handle of the affected interface.
- **`nAddrs`**: the number of entries in the IP address array.
- **`mcAddrArray`**: pointer to an array of IPv4 multicast addresses to remove.

Output Parameters

None

Asynchronous Error Codes

- **`NPF_NO_ERROR`**: Operation successful.
- **`NPF_IF_IPV4_E_INVALID_HANDLE`**: **`if_Handle`** is null or invalid, or is not an IPv4 interface.

Asynchronous Response

A total of **`nAddrs`** asynchronous responses (**`NPF_IfAsyncResponse_t`**) will be passed to the callback function, in one or more invocations. Each response contains one of the given addresses and a success code or a possible error code for that address.

4.3.10 NPF_IPv4TunnelHopsSet: Set IP-in-IP Tunnel Length (Hops)

Syntax

```
NPF_error_t NPF_IPv4TunnelHopsSet(
      NPF_IN NPF_callbackHandle_t   if_cbHandle,
      NPF_IN NPF_correlator_t       if_cbCorrelator,
      NPF_IN NPF_errorReporting_t   if_errorReporting,
      NPF_IN NPF_uint32_t           nHandles,
      NPF_IN NPF_IfHandle_t        *if_Handle,
      NPF_IN NPF_uint8_t            hopcount);
```

Description

This function sets the tunnel length (hop count) on one or more IPv4-in-IPv4 Tunnel interfaces. If multiple interface handles are given, the same length value is applied to each. This value can be used in setting the TTL or Hop Limit field of the outgoing tunnel header.

Input Parameters

- **`if_cbHandle`**: the registered callback handle.
- **`if_cbCorrelator`**: the application's context for this call.
- **`if_errorReporting`**: the desired callback.
- **`nHandles`**: The number of handles in the Interface Handle array.
- **`if_Handle`**: pointer to an array of one or more Interface Handles.
- **`hopcount`**: The tunnel length value.

Output Parameters

None

Asynchronous Error Codes

- **NPF_NO_ERROR**: Operation successful.
- **NPF_IF_IPV4_E_INVALID_HANDLE**: **if_Handle** is null or invalid, or is not a tunnel interface.

Asynchronous Response

One asynchronous response structure (**NPF_IfAsyncResponse_t**) for each interface will be passed to the callback function, each containing an interface handle and a success code or a possible error code. The union in the callback response structure is unused.

4.3.11 NPF_IPv4TunnelDSCP_Set Set IP Tunnel DSCP

Syntax

```
NPF_error_t  NPF_IPv4TunnelDSCP_Set(
      NPF_IN NPF_callbackHandle_t   if_cbHandle,
      NPF_IN NPF_correlator_t       if_cbCorrelator,
      NPF_IN NPF_errorReporting_t   if_errorReporting,
      NPF_IN NPF_uint32_t           n_handles,
      NPF_IN NPF_IfHandle_t        *if_HandleArray,
      NPF_IN NPF_uint8_t           *if_DSCPArray);
```

Description

This function assigns IPv4 DiffServ Code Point value to one ore more IPv4 tunnel interfaces. The if_Handle and if_IPv4DSCP arrays must both contain the same number of entries, equal to the value of n_handles; and an element in the nth position in one array must correspond to the element in the nth position in the other.

Input Parameters

- **if_cbHandle**: the registered callback handle.
- **if_cbCorrelator**: the application's context for this call.
- **if_errorReporting**: the desired callback.
- **n_handles**: the number of interfaces to set the DSCP for.
- **if_HandleArray**: pointer to an array of interface handles.
- **if_DSCPArray**: pointer to an array of IPv4 DSCP values.

Output Parameters

None

Asynchronous Error Codes

- **NPF_NO_ERROR**: Operation successful.
- **NPF_IF_IPV4_E_INVALID_HANDLE**: An **if_Handle** is null or invalid, or is not an IPv6inv4 interface.
- **NPF_IF_IPV4_E_INVALID_DSCP**: A IPv4DSCP is not a valid IPv4 DiffServ code point value.

Asynchronous Response

A total of **n_handles** asynchronous responses (**NPF_IfAsyncResponse_t**) will be passed to the callback function, in one or more invocations. Each response contains an interface handle and a success code or a possible error code for that interface. The union in the callback response structure is unused.

4.3.12 NPF_IfTunnelIPv4AddrSet: Set Tunnel's IPv4 Addresses

Syntax

```
NPF_error_t NPF_IfTunnelIPv4AddrSet(
      NPF_IN NPF_callbackHandle_t   if_cbHandle,
      NPF_IN NPF_correlator_t       if_cbCorrelator,
      NPF_IN NPF_errorReporting_t   if_errorReporting,
      NPF_IN NPF_uint32_t           nHandles,
      NPF_IN NPF_IfHandle_t         *if_Handle,
      NPF_IN NPF_IfTunnelIPv4Addr_t *addrArray);
```

Description

This function sets the source and destination IP addresses on one or more IPv4-in-IPv4 or IPv6-in-IPv4 Tunnel interfaces.

Input Parameters

- **`if_cbHandle`**: the registered callback handle.
- **`if_cbCorrelator`**: the application's context for this call.
- **`if_errorReporting`**: the desired callback.
- **`nHandles`**: The number of handles in the Interface Handle array and the address array.
- **`if_Handle`**: pointer to an array of one or more Interface Handles.
- **`addrArray`**: Array of source/destination IPv4 address pairs, one pair for each interface handle.

Output Parameters

None

Asynchronous Error Codes

- **`NPF_NO_ERROR`**: Operation successful.
- **`NPF_IF_IPV4_E_INVALID_HANDLE`**: **`if_Handle`** is null or invalid, or is not a tunnel interface.
- **`NPF_IF_IPV4_E_INVALID_IPADDR`**: Invalid source or destination IP address given.

Asynchronous Response

One asynchronous response structure (**`NPF_IfAsyncResponse_t`**) for each interface will be passed to the callback function, each containing an interface handle and a success code or a possible error code. The union in the callback response structure is unused.

4.3.13 NPF_IfIPv4RuntimeMTU_Get: retrieve runtime MTU value

```
NPF_error_t NPF_IfIPv4TunnelMTU_Get (
      NPF_IN NPF_callbackHandle_t if_cbHandle,
      NPF_IN NPF_correlator_t if_cbCorrelator,
      NPF_IN NPF_errorReporting_t if_errorReporting,
      NPF_IN NPF_uint32_t nHandles,
      NPF_IN NPF_IfHandle_t *if_HandleArray
      );
```

4.3.13.1 Description

This function returns via a callback, a pointer to an interface MTU value (NPF_uint16_t) containing the current value used by one or more of the indicated interfaces. **This is an optional function. If not implemented, it shall return `NPF_E_FUNCTION_NOT_SUPPORTED` as a synchronous error.**

This function should be supported on IP-in-Ip tunnel interfaces that support PMTU mode, but it may also be supported on other interfaces. If supported on an interface with static MTU, it should return this static MTU, which then also is the runtime MTU.

4.3.13.2 Input Parameters

- **if_cbHandle**: the registered callback handle.
- **if_cbCorrelator**: the application's context for this call.
- **if_errorReporting**: the desired callback.
- **nHandles**: the number of interfaces to get attributes for.
- **if_HandleArray**: pointer to an array of interface handles.

4.3.13.3 Output Parameters

None.

4.3.13.4 Return Codes

- **NPF_NO_ERROR**: Operation successful.
- **NPF_IF_IPV4_E_INVALID_HANDLE**: An **if_Handle** is null or invalid.

4.3.13.5 Asynchronous Response

A total of **nHandles** asynchronous responses (**NPF_IfAsyncResponse_t**) will be passed to the callback function, in one or more invocations. Each response contains an interface handle or a possible error code. If the error code indicates success, the union in the callback response structure contains a pointer to an **NPF_uint16_t** (carrying the MTU value).

5 References

1 NP Forum – Software API Conventions Implementation Agreement Revision 2.0.

6 API Capabilities

This section defines the capabilities of the Interface Management API.

It summarizes the defined APIs and Events and defines the mandatory and optional features.

6.1 Optional support of specific types

The support of any specific type of interface is optional in an implementation. An implementation MAY support exclusively one type of interface, and still claim compliance to the NP Forum Interface Management API.

6.2 API Functions

Function Name	Required?
`NPF_IfIPv4AddrSet()`	Only if IPv4 interfaces supported
`NPF_IfIPv4AddrClear()`	Only if IPv4 interfaces supported
`NPF_IfIPv4FIBSet()`	Only if IPv4 interfaces supported
`NPF_IPv4UC_AddrSet()`	Only if IPv4 interfaces supported
`NPF_IPv4UC_AddrAdd()`	Only if IPv4 interfaces supported
`NPF_IPv4UC_AddrDelete()`	Only if IPv4 interfaces supported
`NPF_IPv4McastAddrSet()`	Only if IPv4 interfaces supported
`NPF_IPv4McastAddrAdd()`	Only if IPv4 interfaces supported
`NPF_IPv4McastAddrDelete()`	Only if IPv4 interfaces supported
`NPF_IPv4TunnelHopsSet()`	Only if Tunnel interfaces supported
`NPF_IPv4TunnelDSCP_Set()`	Only if Tunnel interfaces supported
`NPF_IfTunnelIPv4AddrSet()`	Only if IPv4 Tunnel interfaces supported
`NPF_IfIPv4RuntimeMTU_Get()`	Only if IPv4 Tunnel interfaces supported

6.3 API Events

Event Name	Required?
`NPF_IF_IPV4_MTU_CHANGE`	Yes
`NPF_IF_IPV4_FIB_CHANGE`	Yes

APPENDIX A HEADER FILE: NPF_IF.H

```
/*
 * This header file defines typedefs, constants, and functions
 * that apply to the NPF Interface Management API, support for
 * IPv4 and IPv4 Tunnel interface types.
 */
#ifndef __NPF_IF_IPV4_H__
#define __NPF_IF_IPV4_H__

#ifdef __cplusplus
extern "C"  {
#endif

#define NPF_IF_TYPE_IPV4 5                /* IPv4 logical interface */
#define NPF_IF_TYPE_TUNNEL_IPV4 8         /* IP-in-IPV4 tunnel */

/*
 * IPv4 link capability attribute
 */
typedef enum {
      NPF_IF_IPV4_MULTICAST        = 1,   /* multicast */
      NPF_IF_IPV4_POINT_TO_POINT   = 2,   /* point-to-point */
      NPF_IF_IPV4_NBMA             = 3    /* NBMA */
} NPF_IfIPv4LinkCapability_t;

/*
 *    IP Directed Broadcast Option
 */
typedef enum      {
      NPF_IF_IP_DB_DROP = 1,        /* Drop all directed broadcasts */
      NPF_IF_IP_DB_FORWARD = 2,     /* Enable Forwarding dir. bcasts*/
      NPF_IF_IP_DB_FWD_RCV = 3,     /* Enable forwarding and receive */
      NPF_IF_IP_DB_RECEIVE = 4      /* Enable receipt, no forwarding */
} NPF_IfIpDB_Mode_t;


/*
 *    IPv4 Interface attributes
 */
struct NPF_IfIPv4 {
      NPF_IPv4Prefix_t  addr; /* Primary IPv4 Address and plen */
      NPF_IPv4UC_FwdTableHandle_t  fibHandle; /* Forwarding Info Base*/
      NPF_uint32_t            nUcAddrs;   /* Number of unicast IP addrs */
      NPF_IPv4Prefix_t   *ucAddrs;  /* Array of unicast IP addrs */
      NPF_uint32_t            nMcAddrs;   /* Number of mcast IP addrs */
      NPF_IPv4Address_t *mcAddrs; /* Array of multicast IP addrs */
      NPF_IfIpDB_Mode_t dbMode;     /* Directed Broadcast Control */
      NPF_IfIPv4LinkCapability_t linkCap; /* Link Capability attribute */
};

/*
 * Tunnel interface MTU mode
 */
typedef enum {
```

```
        NPF_IFIPv4_TUNNEL_PMTU_DISABLED = 1,
        NPF_IFIPv4_TUNNEL_PMTU_ENABLED = 2
} NPF_IfIPv4TunnelMTU_t;

/*
 * Readout of value for runtime MTU change event
 */
typedef struct {
        NPF_IfHandle_t    ifHandle;   /* Interface handle  */
        NPF_uint16_t      MTU;              /* runtime mtu value*/
} NPF_IfIPv4RuntimeMTU_ChangeEvent_t;

/*
 * Tunnel DSCP mode
 */
typedef enum {
        NPF_IFIPv4_INNER_DSCP_DISABLED = 1,
        NPF_IFIPv4_INNER_DSCP_ENABLED  = 2
} NPF_IfIPv4InnerDSCP_t;

/*
 *    IP-in-IPv4 Tunnel Interface Attributes
 */
struct NPF_IfTunnelIPv4 {
        NPF_IPv4Prefix_t  primAddr;   /* Primary IPv4 address prefix */
        NPF_uint8_t maxHops;    /* Maximum hops in the tunnel */
        NPF_IPv4Address_t dstAddr;    /* Tunnel destination IP addr */
        NPF_IPv4Address_t srcAddr;    /* Tunnel source IP addr */
        NPF_uint8_t DSCP;             /* DiffServ Code Point for hdr */
        NPF_IPv4UC_FwdTableHandle_t fibHandle; /*FIB for forwarding inner hdr*/
        NPF_IfIPv4InnerDSCP_t IPv4DSCP; /*Use DSCP from inner pck*/
        NPF_IfIPv4TunnelMTU_t MTU_MODE; /* PMTU on/off */
        NPF_uint32_t      nAddr;            /*Number of IPv4 addresses*/
        NPF_IPv4Prefix_t *Addrs;      /*Array of IPv4 addresses*/
        NPF_uint32_t      nMcAddrs;   /* Number of mcast IP addrs */
        NPF_IPv4Address_t *mcAddrs;   /* Array of multicast IP addrs */
        NPF_IfIPv4LinkCapability_t  linkCap; /* Link Capability attribute */
};


/*
 * IPv4 Tunnel Addresses
 */
typedef struct    {
        NPF_IPv4Address_t destAddr;         /* Tunnel destination address */
        NPF_IPv4Address_t srcAddr;          /* Tunnel source address*/
} NPF_IfTunnelIPv4Addr_t;

/*
 *    Completion Callback Types
 */
#define NPF_IF_IPV4ADDR_SET         ((NPF_IF_TYPE_IPV4<<16)+1)
#define NPF_IF_IPV4ADDR_CLEAR       ((NPF_IF_TYPE_IPV4<<16)+2)
#define NPF_IF_IPV4FIB_SET          ((NPF_IF_TYPE_IPV4<<16)+3)
#define NPF_IF_IPV4_UC_ADDR_SET           ((NPF_IF_TYPE_IPV4<<16)+4)
#define NPF_IF_IPV4_UC_ADDR_ADD           ((NPF_IF_TYPE_IPV4<<16)+5)
#define NPF_IF_IPV4_UC_ADDR_DELETE  ((NPF_IF_TYPE_IPV4<<16)+6)
```

```
#define NPF_IF_IPV4_MCAST_ADDR_SET  ((NPF_IF_TYPE_IPV4<<16)+7)
#define NPF_IF_IPV4_MCAST_ADDR_ADD  ((NPF_IF_TYPE_IPV4<<16)+8)
#define NPF_IF_IPV4_MCAST_ADDR_DELETE     ((NPF_IF_TYPE_IPV4<<16)+9)
#define NPF_IF_TUNNEL_HOPS_SET            ((NPF_IF_TYPE_IPV4<<16)+10)
#define NPF_IF_TUNNEL_DSCP_SET            ((NPF_IF_TYPE_IPV4<<16)+11)
#define NPF_IF_TUNNEL_IPV4_ADDR_SET ((NPF_IF_TYPE_IPV4<<16)+12)
#define NPF_IF_IPV4_RUNTIME_MTU_GET       ((NPF_IF_TYPE_IPV4<<16)+13)

/*
 *    Asynchronous error codes (returned in function callbacks)
 */

/* Invalid IP address */
#define NPF_IF_IPV4_E_INVALID_IPADDR      ((NPF_IF_TYPE_IPV4<<16) + 1)

/* Invalid IP  prefix length */
#define NPF_IF_IPV4_E_INVALID_PLEN        ((NPF_IF_TYPE_IPV4<<16) + 2)

/* Invalid IP MTU specification */
#define NPF_IF_IPV4_E_INVALID_MTU         ((NPF_IF_TYPE_IPV4<<16) + 3)

/* Invalid FIB handle */
#define NPF_IF_IPV4_E_INVALID_FIB_HANDLE  ((NPF_IF_TYPE_IPV4<<16) + 4)

/* Invalid TTL value */
#define NPF_IF_IPV4_E_INVALID_TTL         ((NPF_IF_TYPE_IPV4<<16) + 7)

/* Invalid DSCP value */
#define NPF_IF_IPV4_E_INVALID_DSCP        ((NPF_IF_TYPE_IPV4<<16) + 8)

/* Invalid IPv4 Multicast Address */
#define NPF_IF_IPV4_E_INVALID_IPV4_MCAST_ADDR ((NPF_IF_TYPE_IPV4<<16) + 9)


/*
 *    IPv4 and IPv4 Tunnel Event types
 */

/* Runtime MTU changed */
#define NPF_IF_IPV4_MTU_CHANGE ((NPF_IF_TYPE_IPV4<<16)+1)


NPF_error_t  NPF_IfIPv4AddrSet(
    NPF_IN NPF_callbackHandle_t     if_cbHandle,
    NPF_IN NPF_correlator_t   if_cbCorrelator,
    NPF_IN NPF_errorReporting_t     if_errorReporting,
    NPF_IN NPF_uint32_t             n_handles,
    NPF_IN NPF_IfHandle_t          *if_HandleArray,
    NPF_IN NPF_IPv4Prefix_t     *if_IPv4AddrArray);

NPF_error_t  NPF_IfIPv4AddrClear(
    NPF_IN NPF_callbackHandle_t     if_cbHandle,
    NPF_IN NPF_correlator_t   if_cbCorrelator,
    NPF_IN NPF_errorReporting_t     if_errorReporting,
    NPF_IN NPF_uint32_t             n_handles,
    NPF_IN NPF_IfHandle_t          *if_HandleArray);
```

```
NPF_error_t  NPF_IfIPv4FIB_Set(
    NPF_IN NPF_callbackHandle_t     if_cbHandle,
    NPF_IN NPF_correlator_t         if_cbCorrelator,
    NPF_IN NPF_errorReporting_t     if_errorReporting,
    NPF_IN NPF_uint32_t             n_handles,
    NPF_IN NPF_IfHandle_t             *if_HandleArray,
    NPF_IN NPF_IPv4UC_FwdTableHandle_t    if_FIB_Handle);

NPF_error_t  NPF_IfIPv4UC_AddrSet(
    NPF_IN NPF_callbackHandle_t     if_cbHandle,
    NPF_IN NPF_correlator_t   if_cbCorrelator,
    NPF_IN NPF_errorReporting_t     if_errorReporting,
    NPF_IN NPF_IfHandle_t           if_Handle,
    NPF_IN NPF_uint32_t             nAddrs,
    NPF_IN NPF_IPv4Prefix_t     *if_IPv4AddrArray);

NPF_error_t  NPF_IfIPv4UC_AddrAdd(
    NPF_IN NPF_callbackHandle_t     if_cbHandle,
    NPF_IN NPF_correlator_t   if_cbCorrelator,
    NPF_IN NPF_errorReporting_t     if_errorReporting,
    NPF_IN NPF_IfHandle_t           if_Handle,
    NPF_IN NPF_uint32_t             nAddrs,
    NPF_IN NPF_IPv4Prefix_t     *if_IPv4AddrArray);

NPF_error_t  NPF_IfIPv4UC_AddrDelete(
    NPF_IN NPF_callbackHandle_t     if_cbHandle,
    NPF_IN NPF_correlator_t   if_cbCorrelator,
    NPF_IN NPF_errorReporting_t     if_errorReporting,
    NPF_IN NPF_IfHandle_t           if_Handle,
    NPF_IN NPF_uint32_t             nAddrs,
    NPF_IN NPF_IPv4Prefix_t     *if_IPv4AddrArray);

NPF_error_t  NPF_IfIPv4McastAddrSet(
    NPF_IN NPF_callbackHandle_t     if_cbHandle,
    NPF_IN NPF_correlator_t   if_cbCorrelator,
    NPF_IN NPF_errorReporting_t     if_errorReporting,
    NPF_IN NPF_IfHandle_t           if_Handle,
    NPF_IN NPF_uint32_t             nAddrs,
    NPF_IN NPF_IPv4Address_t        *mcAddrArray);

NPF_error_t  NPF_IfIPv4McastAddrAdd(
    NPF_IN NPF_callbackHandle_t     if_cbHandle,
    NPF_IN NPF_correlator_t   if_cbCorrelator,
    NPF_IN NPF_errorReporting_t     if_errorReporting,
    NPF_IN NPF_IfHandle_t           if_Handle,
    NPF_IN NPF_uint32_t             nAddrs,
    NPF_IN NPF_IPv4Address_t        *mcAddrArray);

NPF_error_t  NPF_IfIPv4McastAddrDelete(
      NPF_IN NPF_callbackHandle_t   if_cbHandle,
      NPF_IN NPF_correlator_t       if_cbCorrelator,
      NPF_IN NPF_errorReporting_t   if_errorReporting,
      NPF_IN NPF_IfHandle_t         if_Handle,
      NPF_IN NPF_uint32_t           nAddrs,
      NPF_IN NPF_IPv4Address_t      *mcAddrArray);

NPF_error_t NPF_IfTunnelHopsSet(
```

```
    NPF_IN NPF_callbackHandle_t     if_cbHandle,
    NPF_IN NPF_correlator_t   if_cbCorrelator,
    NPF_IN NPF_errorReporting_t     if_errorReporting,
    NPF_IN NPF_uint32_t       nHandles,
    NPF_IN NPF_IfHandle_t           *if_Handle,
    NPF_IN NPF_uint8_t        hopcount);

NPF_error_t  NPF_IfTunnelDSCP_Set(
    NPF_IN NPF_callbackHandle_t     if_cbHandle,
    NPF_IN NPF_correlator_t   if_cbCorrelator,
    NPF_IN NPF_errorReporting_t     if_errorReporting,
    NPF_IN NPF_uint32_t             n_handles,
    NPF_IN NPF_IfHandle_t          *if_HandleArray,
    NPF_IN NPF_uint8_t           *if_DSCPArray);

NPF_error_t NPF_IfTunnelIPv4AddrSet(
    NPF_IN NPF_callbackHandle_t     if_cbHandle,
    NPF_IN NPF_correlator_t   if_cbCorrelator,
    NPF_IN NPF_errorReporting_t     if_errorReporting,
    NPF_IN NPF_uint32_t       nHandles,
    NPF_IN NPF_IfHandle_t           *if_Handle,
    NPF_IN NPF_IfTunnelIPv4Addr_t   *addrArray);

NPF_error_t NPF_IfIPv4TunnelMTU_Get (
    NPF_IN NPF_callbackHandle_t           if_cbHandle,
    NPF_IN NPF_correlator_t         if_cbCorrelator,
    NPF_IN NPF_errorReporting_t           if_errorReporting,
    NPF_IN NPF_uint32_t             nHandles,
    NPF_IN NPF_IfHandle_t           *if_HandleArray
);


#ifdef __cplusplus
}
#endif

#endif
```

APPENDIX B LIST OF COMPANIES BELONGING TO NPF DURING APPROVAL PROCESS

1135

Agere Systems	IBM	Samsung Electronics
Alcatel	IDT	Sandburst Corporation
Altera	Intel	Silicon & Software Systems
AMCC	IP Infusion	Silicon Access
Analog Devices	Kawasaki LSI	Sony Electronics
Avici Systems	LSI Logic	STMicroelectronics
Azanda Network Devices	Modelware	Sun Microsystems
Cypress Semiconductor	Mosaid	Teja Technologies
Ericsson	Motorola	TranSwitch
Erlang Technologies	NEC	U4EA Group
EZ Chip	NetLogic	Xelerated
Flextronics	Nokia	Xilinx
Fujitsu Ltd.	Paion Co., Ltd.	Zettacom
FutureSoft	PMC Sierra	ZTE
HCL Technologies	RadiSys	
Hi/fn		