IPv4 Control Plane

Design Specification

Control Plane-Platform Development Kit 2.11

March 2004

intel

Contents

Figures

Tables

Revision History

Revision	Description	Date	Author
2.11	Updated for Release 2.11	March 2004	Amit Kaul
2.1	Updated for Release 2.1	December 2003	Amit Kaul
2.0	Updated for Release 2.0	August 2003	Amit Kaul

Part 1: Overview

1 Overview

1.1 Purpose and Scope

This document provides information on the internal design of the IPv4 component of Control Plane-Platform Development Kit (CP-PDK). This includes description and design of the main internal data structures as well as algorithms used within the component.

This document describes the design of the IPv4 component. It does not describe the IPv4 Application Program Interface (API) implemented by this component. For definitions on API, refer to IPv4 API Reference [7].

The intended audience for this document includes:

- Developers implementing and maintaining the IPv4 component
- Test engineers performing Quality Assurance (QA) on the IPv4 component
- Application developers who need a better understanding of the underlying implementation of APIs.
- IPv4 API implementation is the code that implements Network Processing Forum (NPF) IPv4 application APIs. This comprises various subcomponents within control as well as forwarding plane that are responsible for actions such as
    - Validating input
    - Maintaining state per API call , such as, callback data. Refer to [2 for more details on callback model
    - Contacting FP plug-in API to send requests to the forwarding elements (FEs)

1.2 Requirements

IPv4 component must implement the complete set of IPv4 APIs as specified by NPF IPv4 Unicast Forwarding Service API implementation agreement. It must also follow the semantics and behavior outlined in PDK Framework Reference [2].

1.3 Assumptions and Dependencies

The following assumptions are made in design:

- PDK APIs assume the existence of a compliant namespace present on the system. This namespace provides handles to access objects, such as, virtual router and interface.
- A route is made of a prefix and next-hop. In this document, the terms – a route, a prefix, and next-hop, are used interchangeably.

- The design as defined in this document assumes IPv4 application would be able to pass on the table handle that it gets after creating the table using IPv4 NPF API in IPv4 interface management API of `NPF_IfIPv4FibSet`.
- IPv4 API implementation uses FP plug-in APIs to communicate with multiple FEs.
- If there is any error in any sub-operation, that might be part of a bigger operation targeted towards the forwarding plane, an error return status for the main operation is provided to the application.

1.4 Terminology

Table 1 lists terms used in this document and provides an expansion for each term.

Table 1. Terminology table

Term	Description
CE	Control Element
FE	Forward Element
IXA	Internet eXchange Architecture
NPF	Network Processing Forum
PDK	Platform Development Kit
CP	Control Plane
FP	Forwarding Plane
FPP / FPPI	Forwarding Plane plug-in
IA	Implementation Agreement (Approved Draft in NPF)
L2	Layer 2

1.5 References

Table 2 lists documents referenced in, or related to, this document.

Table 2. Reference table

Reference	Document Name
[1]	Forwarding Plane Module – Design Reference
[2]	NPF API Framework Reference

Reference	Document Name
[3]	Software Architecture Overview
[4]	Topology and Label Manager Design Reference
[5]	Namespace Design Reference
[6]	FP Module Core Component Design Reference
[7]	NPF IPv4 API Implementation Agreement

Part 2: IPv4 Component Design

2 IPv4 Component Design

2.1 IPv4 Component High-Level Overview

The primary tasks of IPv4 component NPF API implementations include:

- Validating input from applications
- Maintaining state associated with API c from applications
- Communicating with forwarding elements through FP plug-in.

The API implementation has no knowledge of inter-FE forwarding details.

Any request from the application to IPv4 manager module can be split into three parts:

- Input validation
- Sending request to FP
- Returning a result to the client

In between the validation and route download, it gives data to FP plug-in. After API implementation receives this callback, it can complete the task of downloading routes, or if necessary, indicate an error to the client.

If API implementation has to handle multiple application requests while waiting for several outstanding requests to complete, it must store state and some context for each request to keep track of stage of operation.

The data structures used to store the state for FP plug-in calls and the state required for user application interaction are described in detail in later sections of this document.

2.2 IPv4 Component Block Diagram

Figure 1 depicts relationship between the control plane and the forwarding plane. It also depicts sub-modules within each plane that are related to IPv4 implementation in CP-PDK.

As shown in the figure 1, the CP-PDK IPv4 comprises sub modules as explained in following sub-sections that exist in both control plane as well as forwarding plane.

2.2.1 NPF IPv4 Unicast Service API Interface

The applications written prior to the CP-PDK uses NPF IPv4 unicast service API. This API conforms to latest IPv4 implementation published in the NPF Website.

2.2.2 IPv4 Manager

This sub-module is the core of IPv4 implementation in the CP-PDK and takes care of maintaining context and information related to calls from the user application to the forwarding plane. This sub-component also takes care of interacting with other auxiliary modules in CP-PDK suite. This document concentrates on design and logic applicable in this module.

2.2.3 Transport Plug-in – IPv4 FPPAPI

This sub-module takes care of packing information related to the operation that the user has asked for, from control plane to remote forwarding plane. This part of transport plug-in resides in the control element. If both control and forwarding plane components are co-located, then there is no need for any transport plug-in module.

2.2.4 Transport Plug-in – IPv4 Backend API

This sub-module takes care of unpacking information related to the operation from control plane. This part of transport plug-in resides in every forwarding element that can be the communication target of control element.

2.2.5 Forwarding Modules – IPv4 Module

This sub-component resides in every target-forwarding element. This module has its own state machine and maintains context similar to how it is maintained in IPv4 manager in control plane. It also invokes asynchronous calls into core component infrastructure bundled in each FE, core-component APIs of IPv4 forwarder and other related core components in Software Development Kit.

Refer to FP Module Core Component Design Reference [6] for more details about this module.

2.2.6 Interconnect

When the forwarding plane is at a remote location, communication between the control plane and the forwarding plane, can happen through a variety of interconnect technologies. Intel's offering of CP-PDK abstracts various inter-connect technologies using FORCes.

Details of other auxiliary modules , such as, namespace, PDK manager, L2 manager that are shown in figure 1 are not within the scope of this document. This document lists some basic interactions between IPv4 manager and other auxiliary modules.

2.3 Dependencies

IPv4 manager depends on several other internal PDK components. IPv4 uses namespace and configuration and management to add and find objects and their attributes, such as, FEs and tables. It depends on the interface management module to associate a prefix or FIB table with an IPv4 interface. IPv4 component uses FP plug-in API to communicate with forwarding planes. It also depends on PDK manager to initialize and shutdown.

At present, no other PDK component has dependency on IPv4.

2.4 External API

IPv4 manager implements IPv4 application API as described in NPF IPv4 API Implementation Agreement [7]. Refer to this document for details on external API. As a part of CP-PDK release 2.11, IPv4 module is going to support unified APIs by providing mapping in control plane to the underlying implementation of discrete APIs. NPF IPv4 service API brings this out as explained in Section 2.2.1.

2.5 Internal API

IPv4 components interact with other PDK framework modules using specific internal API interfaces as exposed by the target module Figure

The various PDK framework modules that IPv4 manager depends on are:

- L2ID manager
- Namespace
- PDK manager

Most of these modules describe APIs that start with `pdk_ prefix`.

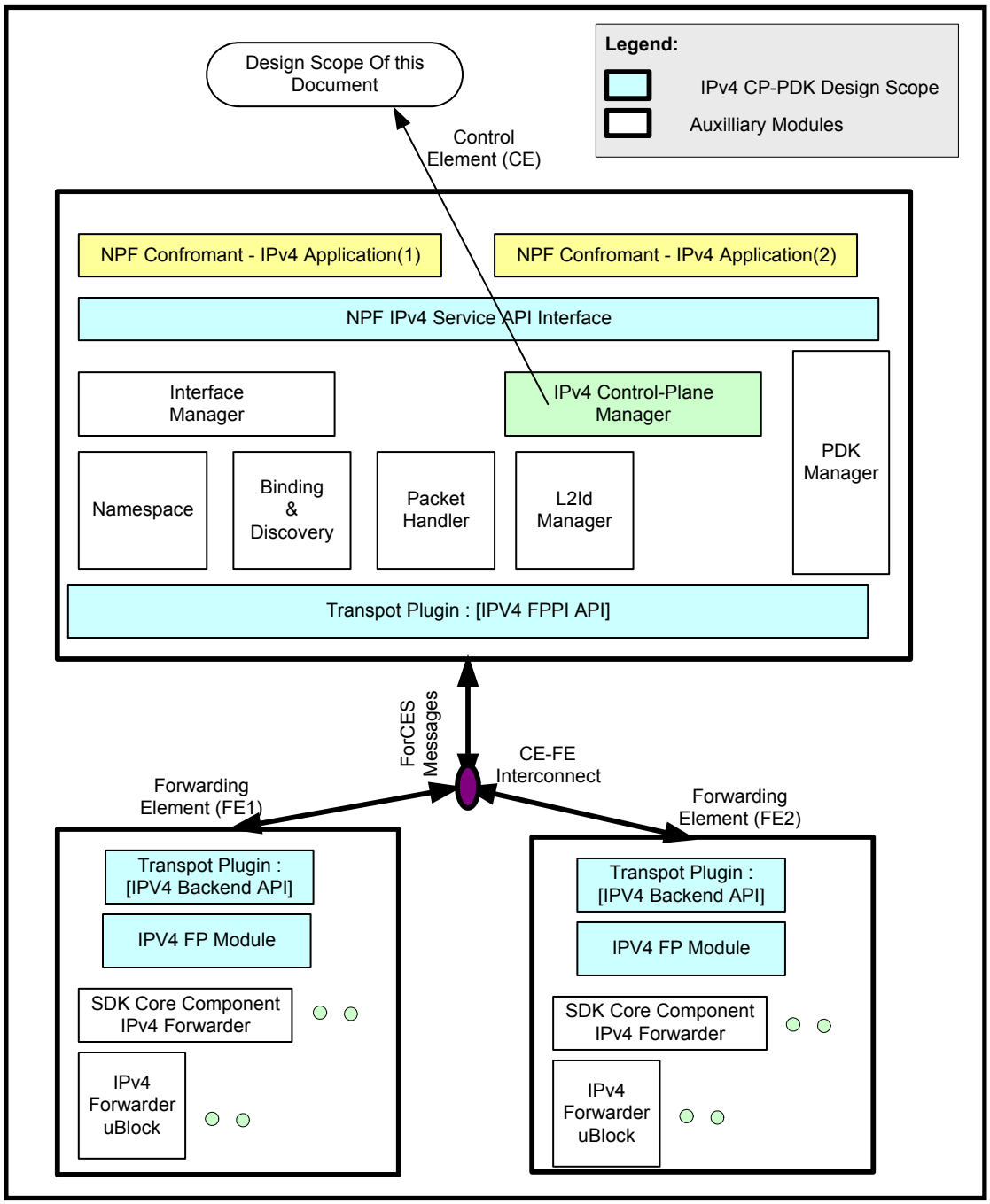


Figure 1. IPv4 component block diagram

Part 3: IPv4 Control Plane Manager

3 IPv4 Control Plane Manager

This section covers the information on design of IPv4 control plane manager.

3.1 Initialization and Shutdown

3.1.1 IPv4 Manager Startup

PDK manager invokes startup routine for IPv4 manager. IPv4 manager uses namespace, C&M, and FP plug-in API, so these components must be initialized before starting IPv4 component.

Startup routine has the following responsibilities:

- Initialize internal data structures , such as, lists to maintain invocation state
- Set initialization flag
- Register callbacks with other internal components
- Register for interface events
- Register for FE events

3.1.2 IPv4 Component Shutdown

PDK manager calls IPv4 shutdown routine. This routine is responsible for releasing resources currently in use and terminating any communication with other internal components. This routine causes all outstanding application callbacks to be invoked to indicate errors to the client applications. Shutdown responsibilities are as follows:

- Reset initialization flag so no additional API calls can be made by applications
- Deregister callbacks with other PDK components
- Invoke all application callbacks to indicate error.
- Release resources , such as, locks and memory

Since IPv4 must deregister callbacks, the CM and FP plug-in API components should be shut down after IPv4.

3.2 Functional Decomposition

The Figure displays various blocks within an IPv4 module that can handle different functionalities of IPv4 control plane manager.

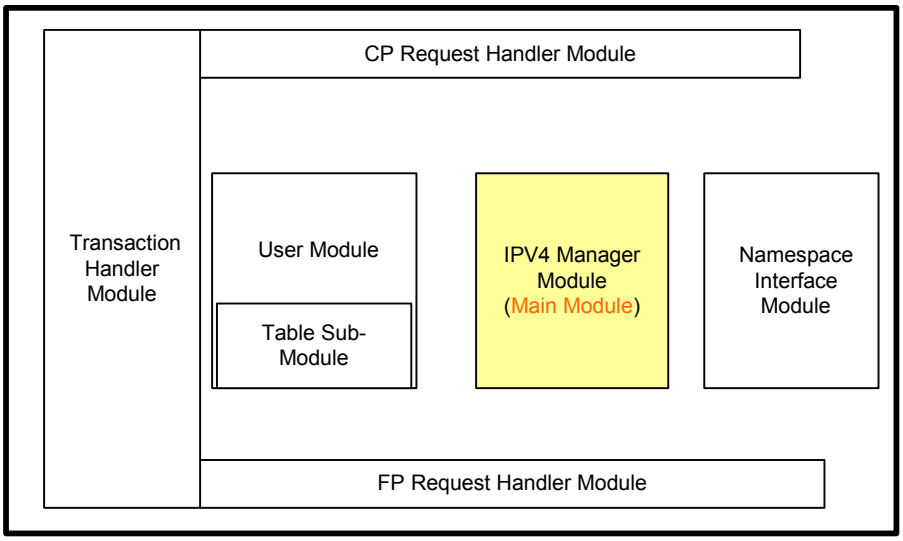


Figure 2 IPv4 control plane manager – block diagram

3.2.1 CP Request Handler Module

This module handles requests that flow from NPF interface module. It is responsible for keeping the context and state of all pending requests to forwarding plane (FP) request handler module. Context of a request from NPF application is information , such as, correlator, verbosity, callback-handle, and table-handle.

3.2.2 FP Request Handler Module

This module handles requests that are to be sent to different FEs using the FEIds. Namespace interface module exposes the API to know the FEId associated with a particular table on which the operation is targeted. This module keeps the pending state for all the requests that are sent to each forwarding element through ForCES messages. An example of context related to request to FP is - FPP correlator and return status.

3.2.3 Transaction Handler Module

This module handles overall transaction for a request as it flows from user application to forwarding plane. If there are multiple FEs as target to an operation, this module has the logic to link one CP request to multiple FP requests in pending state. When one or more FP request is completed, this module has the logic to return valid error or success status to NPF application.

3.2.4 User Module

This module handles various SAPI user registrations with IPv4 module in CP-PDK. It has one more sub-module that handles event user registrations. This has the facility to take care of multithreaded user application. It also holds list of tables that are created within each user context.

This module contains routines that manipulate tables associated with each user. Each user can have a maximum of one FIB table or a pair of next-hop and prefix tables. Keeping this design consideration, if the same NPF application registers with IPv4 module, with two different context values in the registration API, this is equivalent to two user applications handling two different FIB tables for IPv4 manager.

3.2.5 IPv4 Manager Module

This is the main module in IPv4 control plane manager. This module has the module level lock and counters. The module handles initialization and shutdown of IPv4 manager through PDK manager.

3.2.6 Namespace Interface Module

This module is used to interface with the namespace module in control plane. It has the logic to add, update or delete IPv4 tables in namespace using its API. It also has the logic to add or remove associations between various interfaces and FE nodes in namespace tree.

3.3 Data Decomposition

3.3.1 Data-block Breakdown

This section gives details of the various data-blocks in IPv4 control plane manager. Figure 2 illustrates how these blocks are linked and Section 3.3.2 has details regarding each member of the data structure. The following sections briefly explain main data blocks and speak of the rationale behind having these in the design.

IPv4 manager block

The Ipv4 manager block supports the following key features:

- This data block is the primary block that handles the initialization and shutdown requests of IPv4 control plane manager module.
- It contains a lock that is used to protect counters , such as, `fppi_counters` and all other variables or arrays that are within its scope.
- This data block holds the list of user blocks. At initialization time, this list is empty. User blocks are added to this list as and when registration is received from the NPF application (`NPF_IPV4UC_Register`). This data block also holds the list of event user blocks (`NPF_IPV4UC_EventRegister`). Please refer to Section 3.4.1 for more details on this support for multi-user in IPv4 manager design.

- Another important sub-block held by this block is `FPPInfo` block. This block holds all relevant information that is needed for a FP request to be sent to multiple FEs. It also has a list of FP requests that are sent down to FP.

This block is instantiated globally and is accessible to every file.

IPv4 User block

This data block is part of a list that is hived from the main IPv4 manager block. The Ipv4block supports the following key features:

- It is created whenever there is a registration request from user applicaton. Unique key for this data block is the context value that is sent by user application at the time of registration. Whenever there is a new registration, a check is made to make sure that context given by application is not present.
- Another unique key for search is `cbHndl`, that is the value of the callback handle given to the user by IPv4 manager. For every subsequent NPF API request coming from NPF Application after registration, a search is done based on `cbHndl` field in the list of users.
- This block is protected by a lock that is intended to perform the following:
    - Support multithreaded user
    - Protect variables , such as, counters that exist in this block
- This block also holds a list of control plane transactions that in turn contain control plane requests. This is a transient list and lasts from when a request is received from NPF application to when the callback is returned to the NPF application.
- This block also contains an array of the tables created using IPv4 NPF APIs for this user. The reason for keeping these tables within the scope of the user is that all the table creation is done by giving the `cbHndl` as a parameter in those functions, unless the registration has not happened. A table cannot be added without adding a user block. Refer to Section 3.4.2 for further details on the steps in creation of a table.
- For a single user, there can be only one FIB table or pair of prefixes and next hop tables. This requirement is present in NPF IPv4 IA document. As far as design to be implemented in IPv4 manager is concerned, creation of a user is considered as a creation of a router type node. The reason for doing that is because the intent of user application to create a router node is registration. Please refer to Section 3.6.1 for information on steps in this process.

IPv4 Event user block

According to NPF IPv4 IA, such as, registration by user application for IPv4 unicast service API callbacks, user application can also register with IPv4 manager to register the callback handlers for events , such as, `TABLE_EMPTY` from forwarding plane, with a different context value.

Currently, the IPv4 Forwarder core component in SDK 3.5 is not slated to give events as defined in NPF IPv4 API.

Note: As a part of the design exercise for this release of CP-PDK, IPv4 control plane manager does not have functions and related code, except data structures defined in the Figure .

IPv4 Table block

The following are key features of IPv4 table block:

- Stores common table information, such as, state and type in this data block.
- Key to search the array is the handle. This array is fixed in size and depending on whether the APIs being used for the user are unified or discrete, four table blocks or three table blocks are filled. In case of unified APIs, for each FIB table created explicitly by user application, there are two discrete tables created by the unified mapping logic internally, that is prefix table and next-hop table. That makes it three table instances. Fourth table would be the address resolution table that user application will create in both unified and discrete APIs mode.
- This block also contains logic to associate FIB table with the pair of internally created prefix and next-hop tables for unified support.

IPv4 CP Transaction block

The following are key features of IPv4 CP transaction block:

- Unique key for user block is user correlator and internal correlator values. User correlator (`usr_correlator`) stores correlator value that user application sends in each NPF API. Internal correlator (`int_correlator`) holds the current value of the counter (`intCorrelCntr`) present in user block.
- This data block has a type member that can differentiate whether this is a discrete or unified transaction.
- It contains the entire context that is retained when an actual request is received from NPF application , such as, appVerbosity event.
- It contains the gluing logic between control plane request and possible multiple forwarding plane requests going to multiple FEs. This is handled by having pointers to FP requests, and counters to check whether all the requests to FP were returned back.
- In case of the unified mapping logic, this data block is created when the unified API request comes from the user application. It contains logic to have an array of pointers to internally create discrete transaction blocks for quick access. Please refer to Section 3.5.2 for further details on this logic.

3.3.2 Data Structures Linkage

Due to the asynchronous nature of PDK implementation, it is necessary to cache state for API calls and to keep track of registered callbacks. In addition, the common callback library is used to invoke application callbacks. Please refer to NPF API Framework Reference [2] for more details.

Figure depicts the broad level data-structure linkage in the IPv4 manager. Please refer to Section 2.2.2 for more information. This section does not give any design details about other sub-modules , such as, IPv4 FP-plug-in API, IPv4 Backend API and IPv4 module in forwarding plane. Please refer to Figure to check how these modules fit into the whole IPv4 System design.

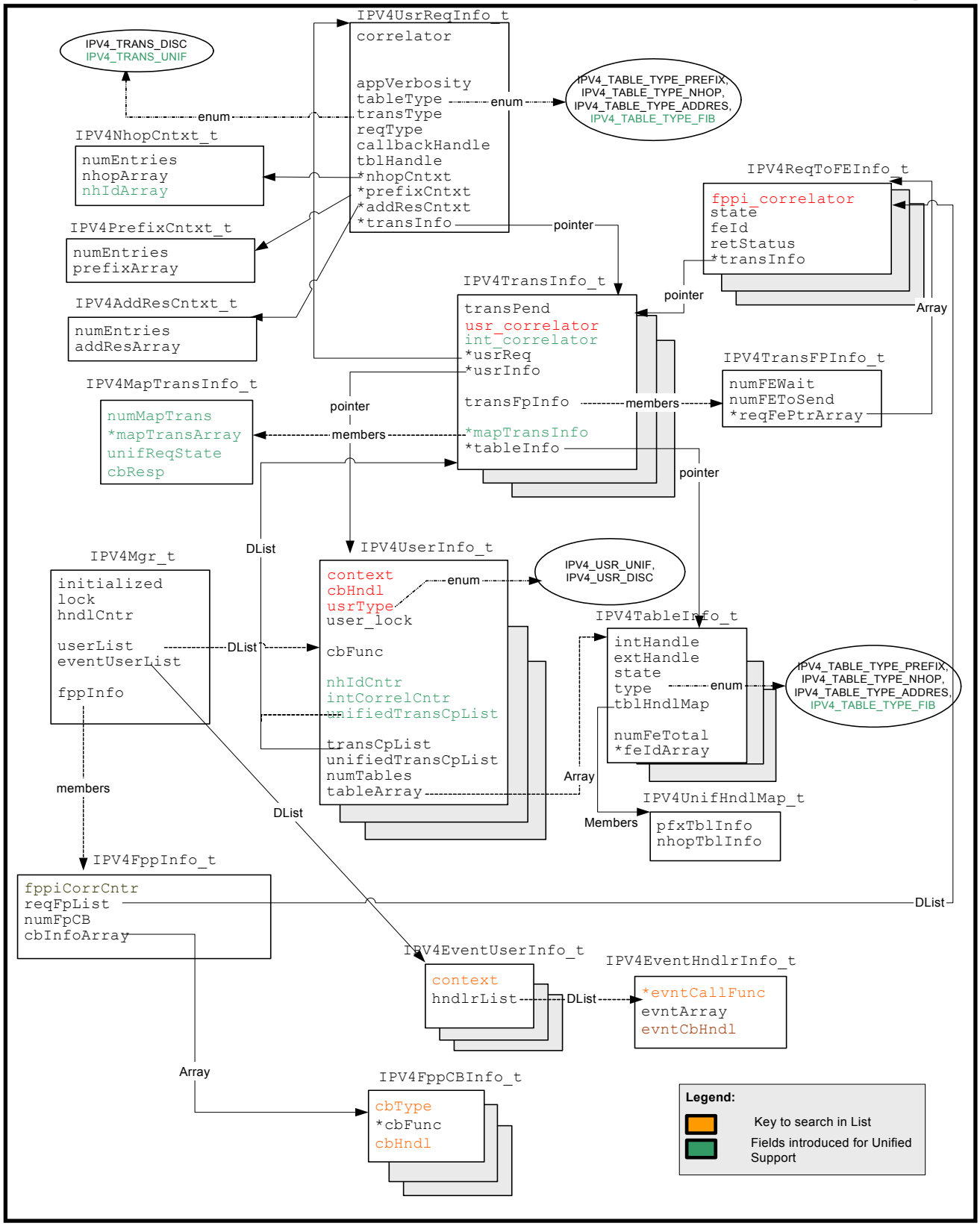


Figure 3 Data Structure Linkage in CP-PDK IPv4 module

IPv4Mgr_t

Table 6. IPv4Mgr_t table

Member	Description
Initialized	TRUE: IPv4 manager has already been initialized FALSE: IPv4 manager has not been initialized
Lock	This lock is used to protect all counters and other unprotected variables that are within the scope of this structure
HndlCntr	A global counter that is incremented each time any user application has to be given a handle (*Handle = hndlCntr++*)
UserList	DList of data structures that have context for each user application, such as, IPV4UserInfo_t, that registers callbacks for NPF IPv4 service APIs
EventUserList	DList of Data Structures that have context for each user application, such as, IPV4EventUserInfo_t, that registers callback for NPF IPv4 events
fppInfo	This structure contains FPP specific information that is accumulated once in PDK startup

IPV4UserInfo_t

Table 7. IPv4UserInfo table

Member	Description
Context	The unique value that the user application provides to the IPv4 manager during registration
cbHndl	The handle that IPv4 module returns back to the user application on registration
usrType	Describes the option that the user application has chosen. It can be: IPV4_USR_UNIF IPV4_USR_DISC The value in this field is set based on whether the user application sends NPF_IPv4UC_CreatePrefixTable (IPV4_USR_DISC) or NPF_IPv4UC_CreateFibTable (IPV4_USR_UNIF). This value cannot be reset once it is set.
user_lock	This lock takes care of multi-threaded user application trying to access various structures hiving from this structure
cbFunc	This is the callback function that user registers with IPv4 manager
nhIdCntr	Next-hop ID counter used to generate NH-IDs internally, in case of mapping from unified to discrete next-hop routines This field in only valid in case of unified mapping
intCorrelCntr	This is a counter used to assign internal correlator to the discrete requests that are triggered by a Unified API

	This field in only valid in case of unified mapping
unifiedTransCpList	Holds the list of transactions that are created in case unified API requests are sent to CP-PDK Ipv4 module.
transCpList	Holds the list of transactions that are created in case discrete API requests are sent to CP-PDK IPv4 module.
numTables	Specify the maximum number of tables. CPPDK IPV4 supports three tables: Prefix, Next-Hop and Address Resolution tables per user in case the mode is discrete. For unified mode, one extra table would be FIB table along with three discrete tables to take care of mapping. Default value = 4 .
tableArray	This is an array of tables per user

IPV4FppInfo_t

Table 8. IPv4FppInfo table

Member	Description
fppiCorrCntr	Counter used to assign FPPI correlator values to the requests going from IPv4 manager towards FPPI
reqFpList	DList of the requests that are targeted towards forwarding plane
numFpCB	Number of forwarding plane callbacks that are to be registered. Value of this decides the size of cbInfoArray field
cbInfoArray	An array of callbacks registered with FPPI, and each node has information , such as, the callback function and the callback handle returned by FPPI (*IPV4FppCBInfo_t*)

IPV4EventUserInfo_t

Table 9. IPV4EventUserInfo_t table

Member	Description
Context	Similar to the IPV4UserInfo_t context member variable, this member is also used to identify a user application that wants to handle events from forwarding plane
hndlrList	DList of event handlers that are registered with the forwarding plane IPv4 module

IPV4EventHndlrInfo_t

Table 10. IPV4EventHndlrInfo_t table

Member	Description
evntCallFunc	Pointer to the callback function registered for handling the events
evntArray	Array of events that the user application has registered for
evntCbHndl	Handle that is returned back to the user application

IPV4Table_t

Table 11. IPV4Table_t table

Member	Description
inthandle	A Handle that is returned to the user application, on creation of the table. The user application must use this handle, whenever it needs to communicate with IPV4 Module. For example, add, delete, query entries.
Exthandle	This handle is returned to the user application only on creation of the prefix and FIB table. This handle is an external handle and the user application should use to communicate with IM module while calling IfIPv4FibSet. Please refer to Section 3.2.6 for further details.
State	This variable holds the current state of the table. Please refer to Section 3.6.2 for further details
Type	This variable gives information on type of table this structure holds IPv4_TABLE_TYPE_PREFIX IPv4_TABLE_TYPE_NHOP IPv4_TABLE_TYPE_ADDRES Ipv4_TABLE_TYPE_FIB
tblHndlMap	This IPV4UnifHndlMap_t contains a pair of handles, each pointing to internally created prefix and next-hop table respectively
numFeTotal	NumFetotal contains the total number of FEs that this table is associated with. The IPV4 module gets this information from the namespace after the IfIPv4FibSet is done by the user application.. If the interfaces passed in this APIs belong to multiple FEs, then this table is also associated with multiple FEs. Currently, the FP support is only for single FIB table for single FEs.
feIdArray	This is an array of feIds that is equal to the numFeTotal.

IPV4TransInfo_t

Table 12. IPV4TransInfo_t table

Member	Description
transPend	There are two conditions: true and false. True: Transaction is pending and callback to the user application cannot be returned False: Transaction is over and if there is any callback to be returned to the user application, it can be given
usr_correlator	This has the value of the user correlator that the user application provides per transaction. For unified API, the meaning of this is different. Please refer to Section 3.5.2 for further details.
int_correlator	This variable stores internal correlator value for discrete API requests that are generated by IPv4 manager internally, in response to one unified API from user application. Please refer to Section 3.5.2 for further details and exact usage semantics.

	In case of a discrete call, value of this is 0. It is applicable only for Unified Mapping logic
usrReq	This member holds user application information and context for request that has to be used when the callback to the user application is given.
usrInfo	This is a back pointer to IPV4UserInfo_t structure for backward reference
transFpInfo	This is IPV4TransFpInfo_t that contains FPP related information and statistics for a transaction to FPP. This information helps in defining logic of collating multiple FP requests for a single CP user Application request.
tableInfo	Pointer to the table for that this transaction is active. If the table is deleted by the user application when there are pending transactions, this field is used.

IPV4UsrReqInfo_t

Table 13. IPV4UsrReqInfo_t table

Member	Description
Correlator	This field holds the correlator that the user application sends along with the NPF API request

appVerbosity	This variable holds the report of whether the user application wants a callback or not
	
	`NPF_REPORT_ALL = 1`
	`NPF_REPORT_NONE = 2`
	`NPF_REPORT_ERRORS = 3`

tableType	Depending on the API type, the user request handler module fills in the right table type. For example, when a request like `NPF_IPv4UC_AddPrefixEntry` is received, the table type is assigned to `IPV4_TABLE_TYPE_PREFIX`
	
	Possible options are:
	
	`IPV4_TABLE_TYPE_PREFIX`
	`IPV4_TABLE_TYPE_NHOP`
	`IPV4_TABLE_TYPE_ADDRES`
	`IPV4_TABLE_TYPE_FIB`

transType	The transType field is filled in depending on the type of API that is received from the user application, which can be one of the following: IPV4_TRANS_DISC IPV4_TRANS_UNIF
reqType	This field holds the type of request that is received. Possible values are taken from the `NPF_IPv4UC_CallbackType_t`.
nhopCntxt	This is the temporary context that is stored in this request when it is targeted to NextHop table. This context is used while interacting with the L2Id manager for L2Id generation and updates.
prefixCntxt	This is the temporary context that is stored in this request when it is targeted to the FIB table. This context is used only in case of unified APIs.
addResCntxt	This is the temporary context that is stored in this request when it is targeted to the address resolution table. This context is used while interacting with the L2Id manager for L2Id

	generation and updates.
transInfo	This is a pointer to the transaction data structure.

IPV4ReqToFEInfo_t

Table 14. IPV4ReqToFEInfo_t table

Member	Description
fppi_correlator	This variable holds the FPPI correlator that is used to send the request to the FPPI
State	This field holds the current state of the request to the FE. Possible values are: REQ_TO_FE_PENDING REQ_TO_FE_SENT REQ_TO_FE_CBACK_RECVD
FeId	FE ID of the target FE for this request
retStatus	Return status – error/success
transInfo	Back pointer to the transaction to which this belongs

IPV4TransFPInfo_t

Table 15. IPV4TransFPInfo_t table

Member	Description
numFEWait	Number of FEs from which we have not received the callbacks. Please refer to Section 3.8 for further details
numFEToSend	Number of FEs to which this request is to be sent.
reqFePtrArray	This contains the array of pointers to the actual FP requests

3.4 Discrete API Design Details

3.4.1 IPv4 Initialization and Multi-user Support

This section explains steps in initialization of IPv4 manager and steps that explain how multiple user applications register with different context values.

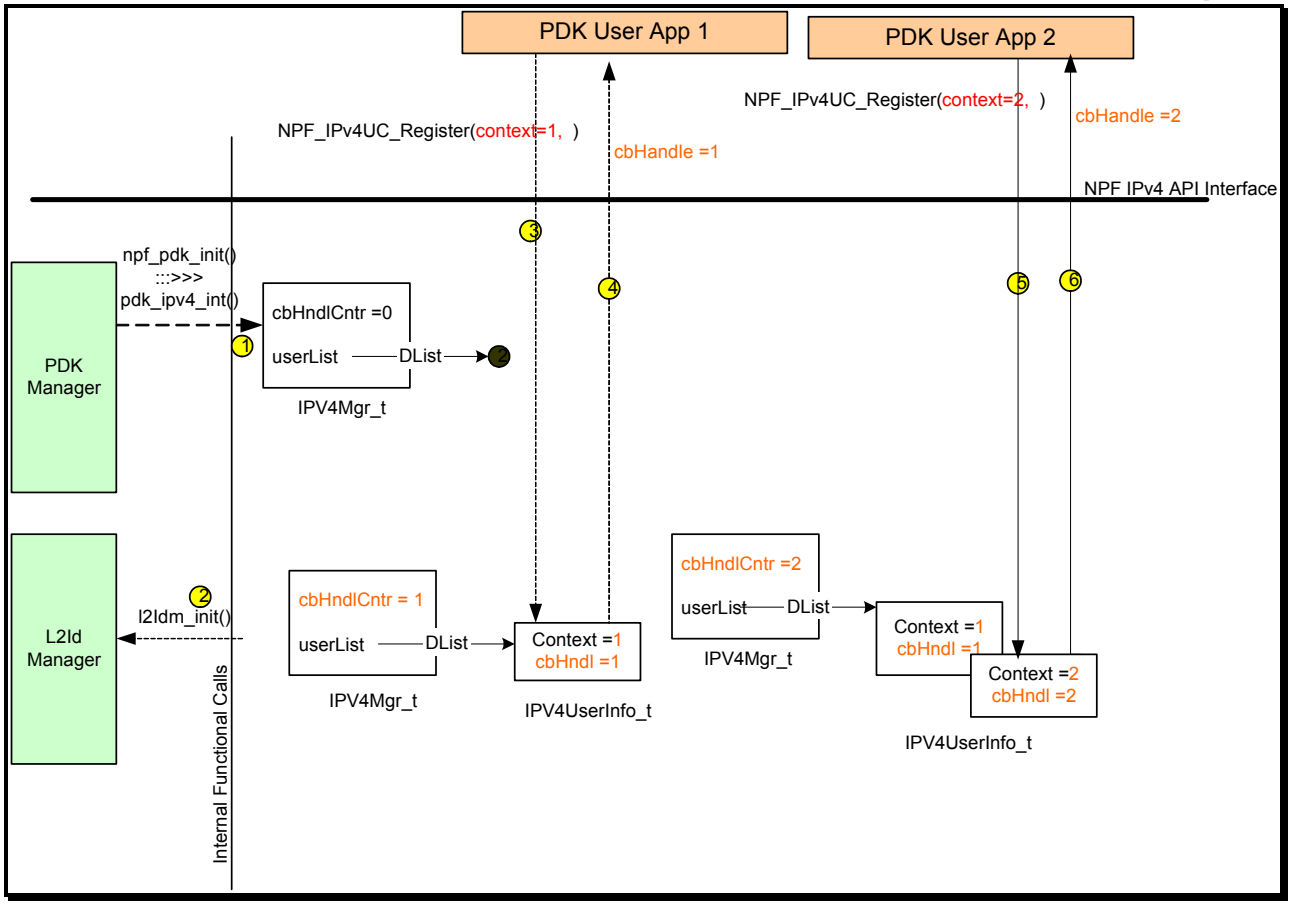


Figure 4 Initialization and multi-user support of IPv4 module

1. PDK manager calls initialization routine of IPv4 module that is `pdk_IPv4_init()`.This creates the main structure `IPV4Mgr_t` and initializes other member variables accordingly.
2. Within `pdk_IPv4_init()`, the L2Id manager is initialized by calling `l2idm_init()`.
3. First user application registers by calling `NPF_IPv4UCRegister` with `context =1`. This is a unique key for this user that is stored inside IPv4 manager by creating a new structure `IPV4UserInfo_t` and adding it to DList from the main structure `IPV4Mgr_t`.
4. From the callback counter `cbHndlCntr,` the handle is given back to the user application. This handle information is also stored in `IPV4UserInfo_t` structure just created that is `cbHandle = 1`
5. Repeat Step 3 with `context = 2`
6. Repeat Step 4. Here `cbHandle = 2`

3.4.2 Discrete API Flow Example – Create Prefix Table

This section explains a simple API flow from the user application to IPv4 Manager.

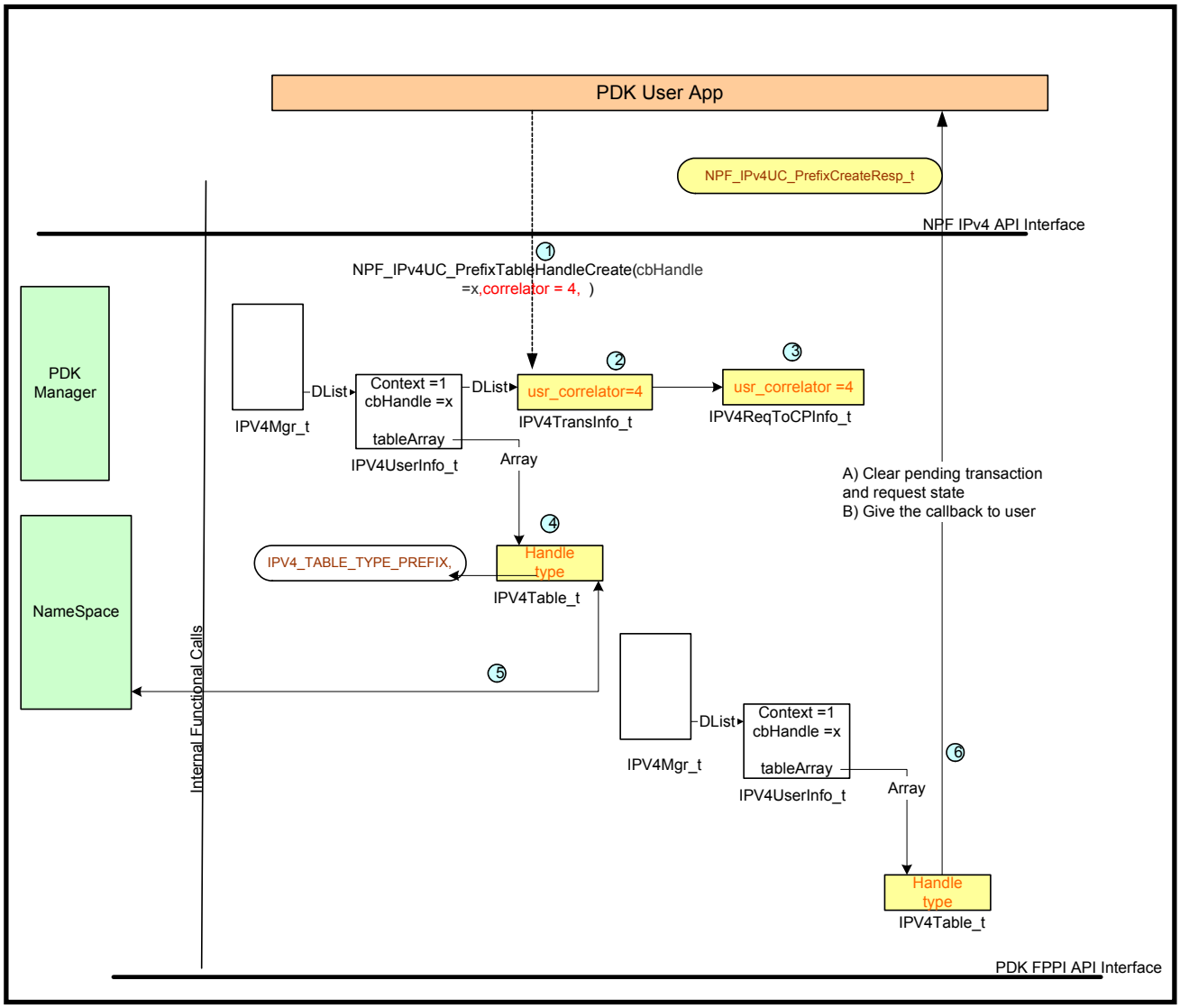


Figure 5 NPF API flow from user application- example

1. Request to create a prefix table is received. User application sends the correlator for this transaction as 4 shown in Figure . After searching based on `cbHandle = x`, `IPV4UserInfo_t` structure is looked for.
2. A check is conducted for any transactions already pending with this correlator. If the result is yes, then return error. If the result is no, then a new transaction structure `IPV4TransInfo_t` is created.
3. New `IPV4ReqToCP_t` that stores all needed context for this request.
4. Check is done in `IPV4UserInfo_t` structure if any table of the same type is present. In this release of CP-PDK, more than one table of the same type is not allowed for a single user. If the table is already present, an error is returned. If the table is not present, then create `IPV4TableInfo_t` structure.
5. There are some interactions with namespace that are explained in Section 3.6.1, in order to get the handle for this table. Please refer to Section 3.6.1for more information.

6. Both `IPV4TransInfo_t` and `IPV4ReqToCP_t` structures, known as transient structures, are deleted after giving the callback with relevant status back to the user application.

3.5 Unified API Design Details

For this release of CP-PDK, the IPv4 module conforms to IPv4 unicast service API implementation agreement. As a part of this agreement, mandatory support has to be provided for unified support. Since it does not support unified support, supporting single FIB table instead of two different tables for prefix and next-hop table, the mapping support has to be provided in IPv4 manager that maps unified APIs to discrete APIs.

Following are the requirements for unified support:

- User application is transparent to any mapping functions provided by IPv4 component.
- NHID in case of discrete next-hop APIs is to be provided by user application. In case of unified, this is generated within the layer. This results in a new NHID being generated whenever a new FIB entry addition is requested. This can result in increase in size of the routing table in forwarding plane.
- Every time there is a delete request for a FIB entry, it would result in a query to the forwarding plane and back to control plane to fetch the valid NHID from forwarding plane. This would result in performance hit.
- In case any discrete API resulting from a unified call fails and other call succeeds, the return value for the unified would be error. Application cannot get exact error value in these cases.

3.5.1 Mapping Details

This section provides information on the mapping logic between unified and discrete calls. This mapping is defined statically in the implementation.

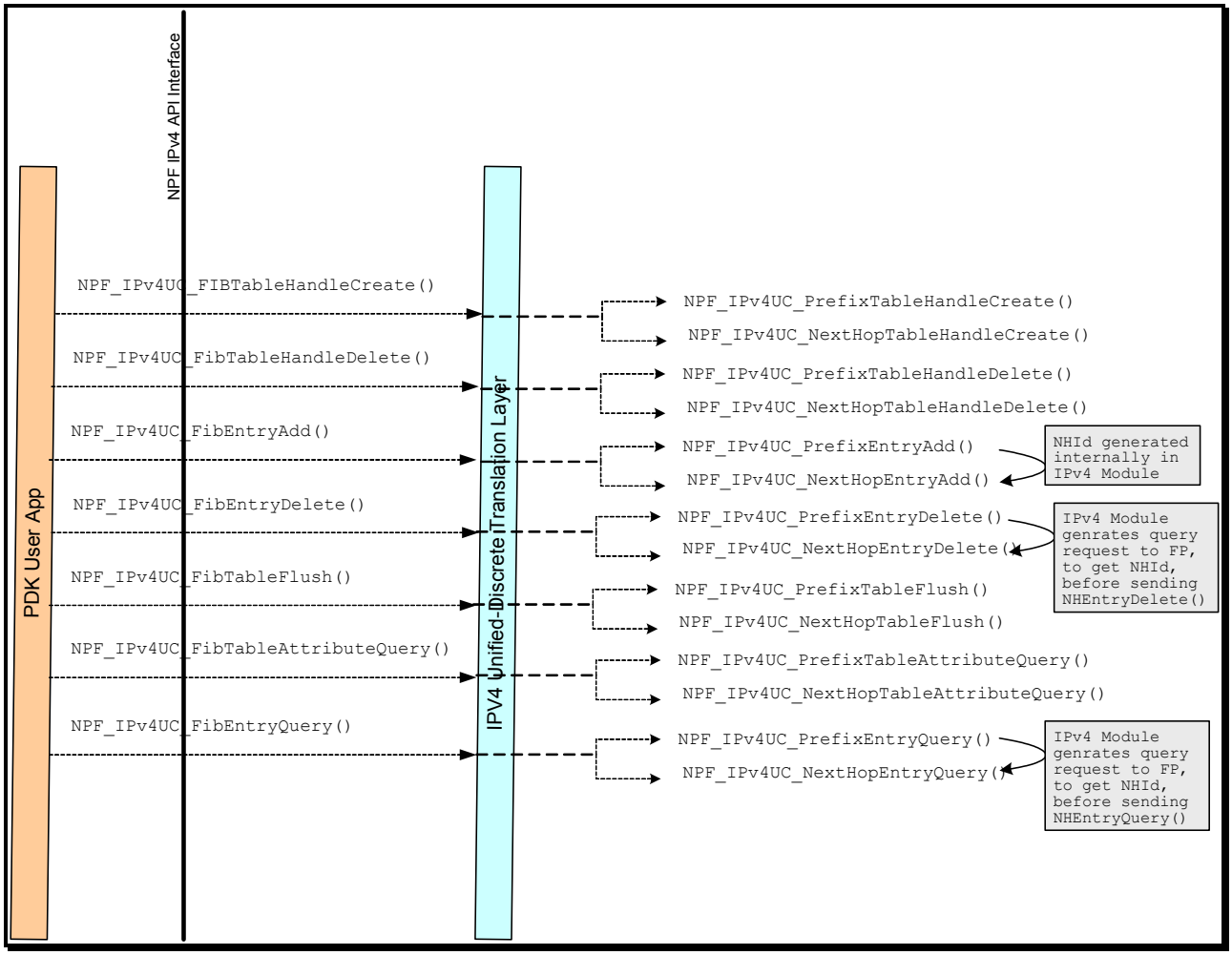


Figure 6 Mapping between unified and discrete calls

3.5.2 Unified API Flow – Example

This section defines an example flow of a unified API call resulting in multiple discrete calls towards forwarding plane. It also defines the creation and deletion of data structures during the flow.

Figure depicts flow for `NPF_IPv4UC_FibEntryAdd ()` and data structure creation/deletion in this case. It assumes that target tables are already created and the associations are already established between FIB table and prefix, next-hop tables.

Following are the steps as defined in the figure 7.

7. `NPF_IPV4UC_FibEntyAdd()` is received by IPv4 manager with correlator = 4 and tableHandle = y.
8. From `UserContext = x, IPV4UserInfo_t` structure is identified. `IPV4TransInfo_t` structure is created with `usr_correlator = 4, int_correlator = 0, transType = IPv4_TRANS_TYPE_UNIFIED.`

9. From tableHandle = y, table `IPV4Table_t structure` is identified from `IPV4UserInfo_t` structure. This table should already have been created and has prefix and next-hop tables with table handles as n and m associated. After verifying that the association is proper and taking the two discrete table handles, the two counters `nhIdCntr` and `intCorrelCntr` in `IPV4UserInfo_t` structure are incremented.
10. Taking the handle values and the counter values, two new transactions are created for generating two new discrete requests internally. These two transactions have the same `usr_correlator = 4` as the one from which they originated. They have new correlator values generated from `intCorrelCntr` assigned to other variable called `int_correlator`. While generating the actual discrete API request, both the correlators are checked. If `int_correlator` is set, then it is given preference over `usr_correlator`. A check is done that these two transactions are of type IPv4_TRANS_DISCRETE.

The association between these three transactions can be detected using the fact that all of these have the same `usr_correlator = 4` value. For quick access, the transaction that is of type `IPv4_TRANS_UNIFIED` has pointers to the mapped transactions. On getting response for all the mapped transactions, the transPend field in unified transaction is false.

11. Taking table handle values further, and the correlator values from `int_correlator`, two APIs are selected from the mapping already defined for `FibEntryAdd` that is `PrefixEntryAdd` and `NextHopEntryAdd`.
12. `PrefixEntryAdd` is generated with relevant values
13. `NextHopEntryAdd` is generated with relevant values. After this step, the flow is similar to the flow of discrete API implementation.

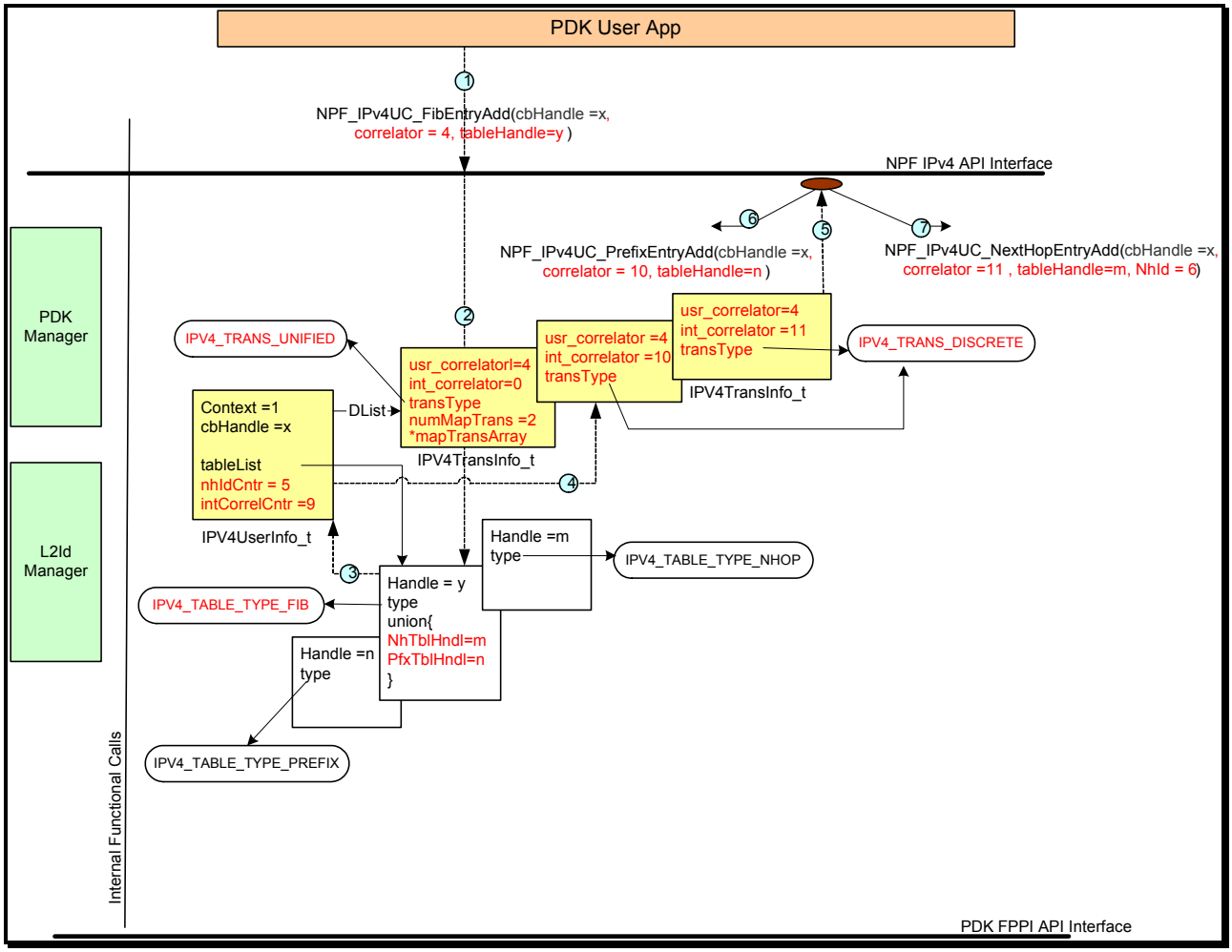


Figure 7 Unified API flow – example

3.6 Common Design Considerations

3.6.1 IPv4 Manager and Namespace Interaction Details

This section gives brief details about how IPv4 manager interacts with namespaces while creating tables. This section acts as a requirement section for changes to be made in IPv4 interface management module.

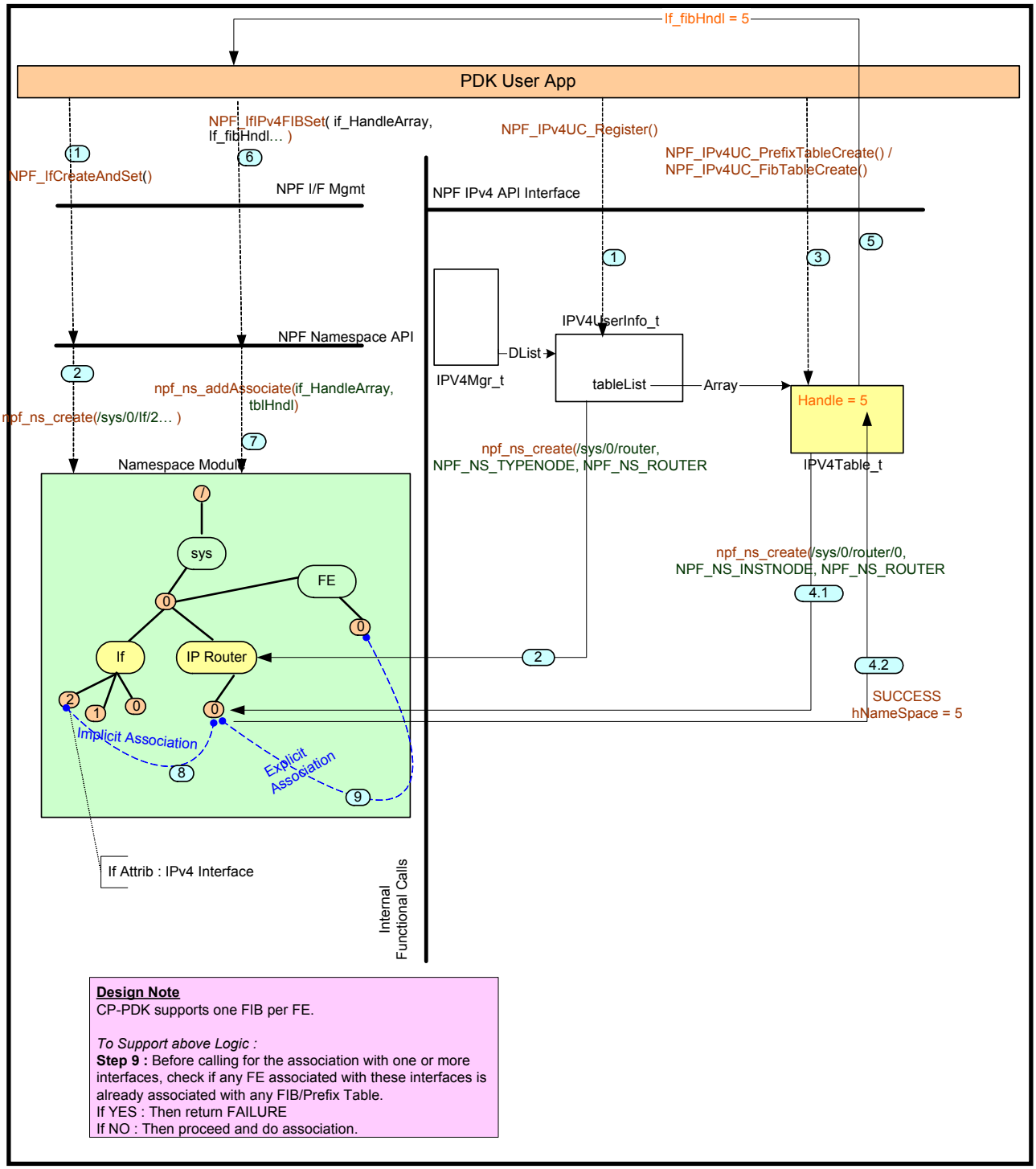
As shown in the Figure , there are two interfaces at the top that the user application has to use in order to create and set a FIB/Prefix table.

These two interfaces are:

- NPF IPv4 unicast Service API
- NPF IPv4 interface management API

It is recommended that the user application follow the steps in the order as mentioned in the Figure 8 as the IPv4 unicast SAPI has API to create a table. The API to set this table to the right IPv4 interface is not available within IPv4 unicast SAPI but it is available in IPv4 interface management API.

At the time of writing this document, the IPv4 interface management document has FIB table handle as its parameter when associating the table with the interface. It does not explicitly expect prefix table handle `typedef` in its parameter. Since, the fundamental data-type of all the handles is `uint32`, it is assumed that prefix table handle can be used in place of FIB table handle argument. Using the same assumption, namespace handle has been passed as prefix/FIB table handle to user application. If, there is a change in the data-type of namespace handle, then there should be an explicit conversion and mapping logic between these two.



Design Note
CP-PDK supports one FIB per FE.

To Support above Logic :
Step 9 : Before calling for the association with one or more interfaces, check if any FE associated with these interfaces is already associated with any FIB/Prefix Table.
If YES : Then return FAILURE
If NO : Then proceed and do association.

Figure 8 IPv4 manager and namespace interaction

3.6.2 Table Finite State Machine

This section provides information on various states of a table for example prefix, next-hop, FIB, and ARP request tables. It also gives details about various events/triggers that result in transition from one state to another.

In this table, there are a few errors that are not implemented in the SDK For example, when we call `AddEntries()` and if the table is full, the IPv4 Forwarder CC should return `TBL_FULL` error. Until this is implemented, FSM does not go to `TABLE_FULL` state.

At the time of prefix or FIB table creation, when the state becomes `TABLE_EMPTY`, the IPv4 manager interacts with the namespace module to create an entry. At the time of their deletion, when the state goes to `TABLE_DELETED`, the IPv4 manager again interacts with namespace module to delete the entry.

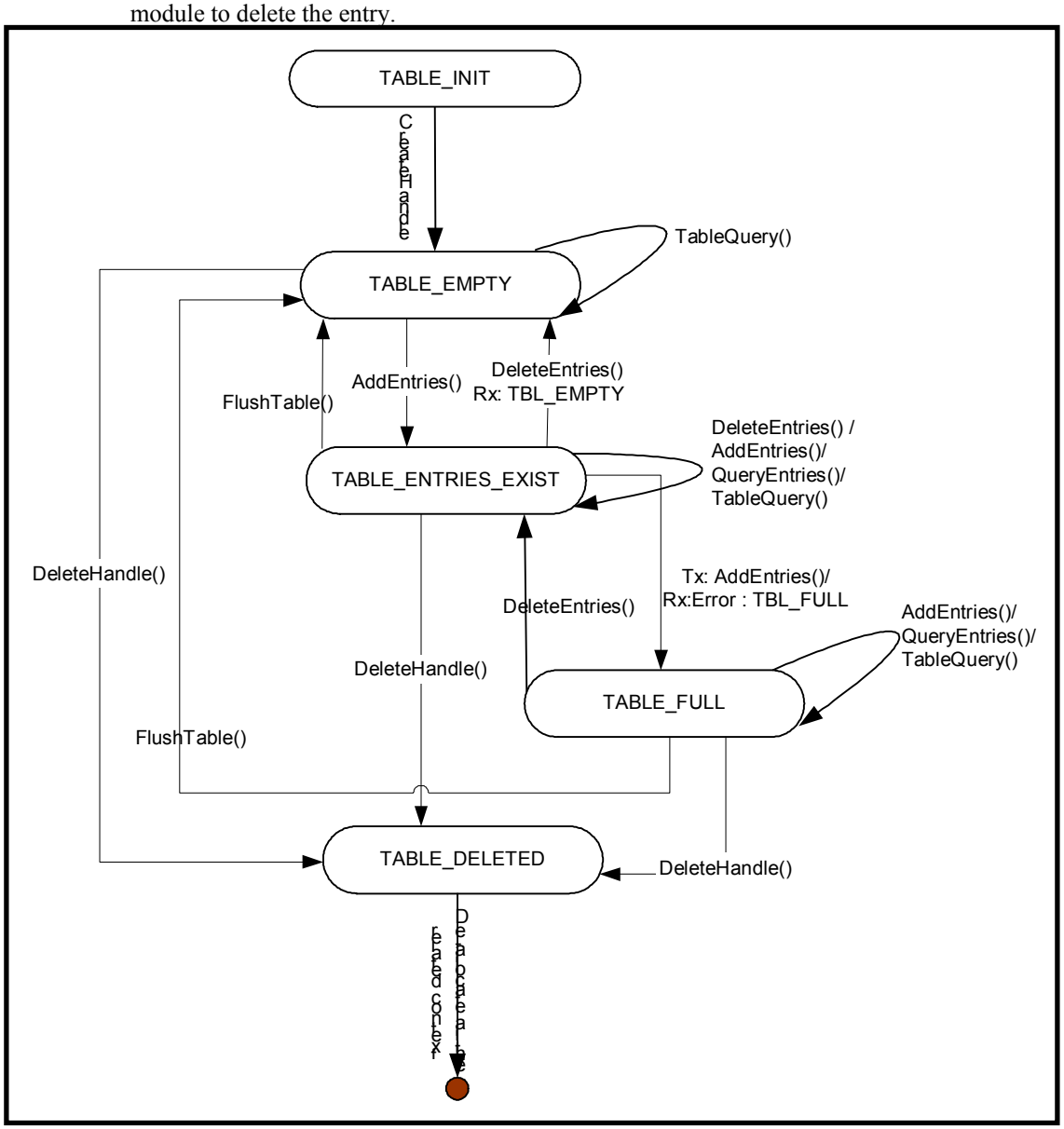


Figure 9 FSM table

3.6.3 IPv4 Mgr – Request Finite State Machine

This section details the Finite State Machine (FSM) of a request from the point it is received from the user application till callback for the request is given back to the user application or, the state is removed immediately after the request is given to the FPP module.

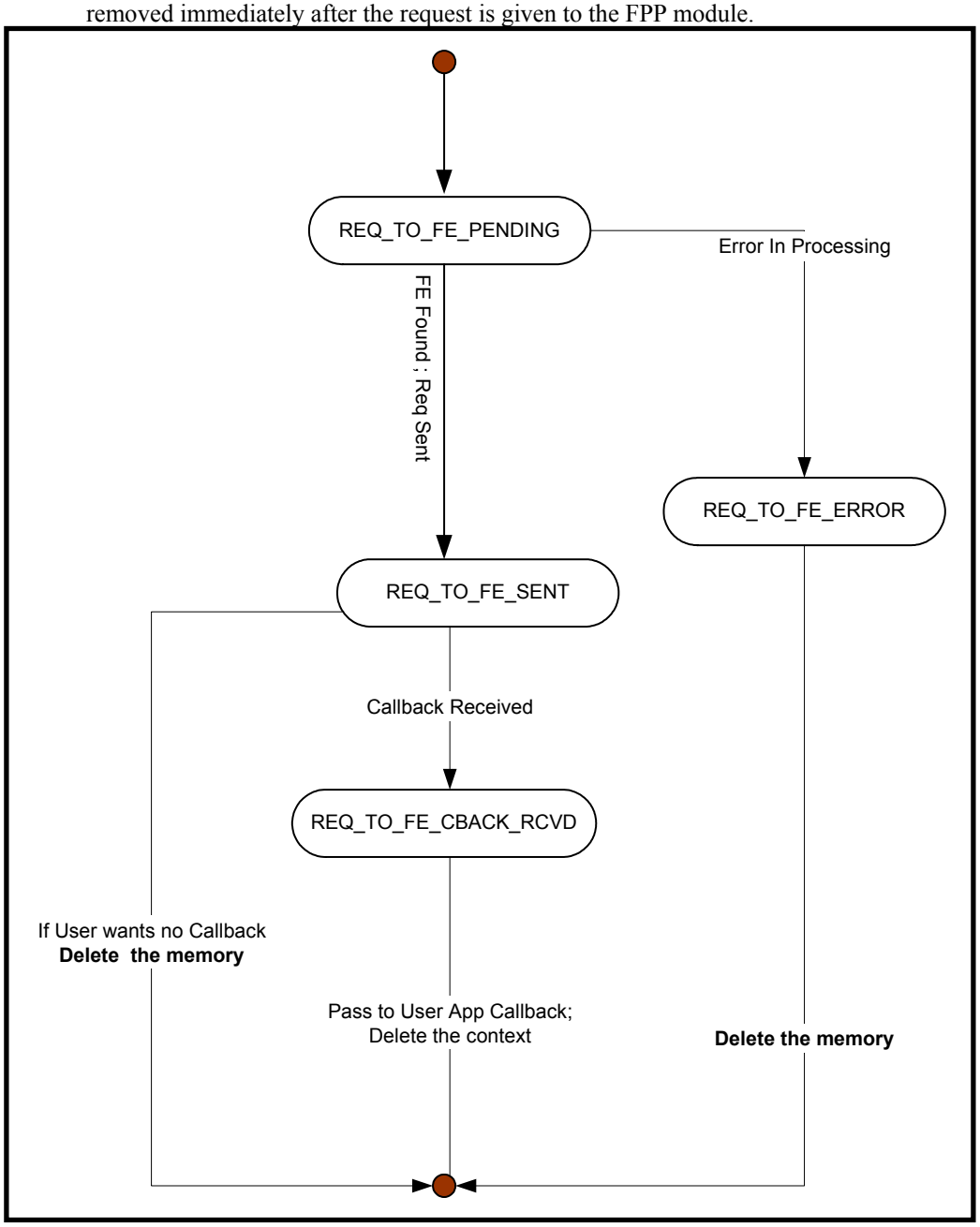


Figure 10 IPv4 NPF API request FSM

3.6.4 Locking

IPv4 Component uses coarse-grained locking. There is a mutex for the main IPv4 module, IPv4 manager acquired for all the cases where the variables have to be modified, for example, incrementing counters, assigning pointers. When IPv4 calls into the FP plug-in, it must first release the global lock.

There is a lock for each IPv4 user to make sure that if there is a multithreaded user application, it can work fine. This lock, , such as, the IPv4 manager lock is invoked and released for the duration during which any variable is incremented or decremented within the scope of user.

For every DList structure in IPv4 control plane manager, there is a lock that is invoked by DList library. The library protects accessing DLists and IPv4 does not have its own specific locks for DList operations.

3.6.5 Directly Connected Hosts

In release 1.2, the logic for special handling of direct connected hosts existed inside the IPv4 Module. In the release 2.11, the events are given to the user application and no special processing is done inside the CP-PDK. It is left to the user application to check for the missing routing table and add the relevant routes using the IPv4 SAPI.

3.6.6 IXA SDK-Specific Design

The following elements are necessary to be compatible with an IXA SDK-specific data plane. They can be easily replaced or extended to support other data planes.

3.6.6.1 L2Id Generation

When one or more next-hops are added to IXA SDK implementation, a unique ID, L2ID, is passed for each FEID/next-hop address pair. Forwarding element next-hop tables maintain a mapping between next-hop identifiers, next-hop addresses, L2Ids, and FEIDs. L2ID is used as a mapping to next-hop L2 addresses in the egress L2 table. NHIDs are used as a mapping to a network destination address. To illustrate L2ID usage, consider the hypothetical network arrangement in the following figure:

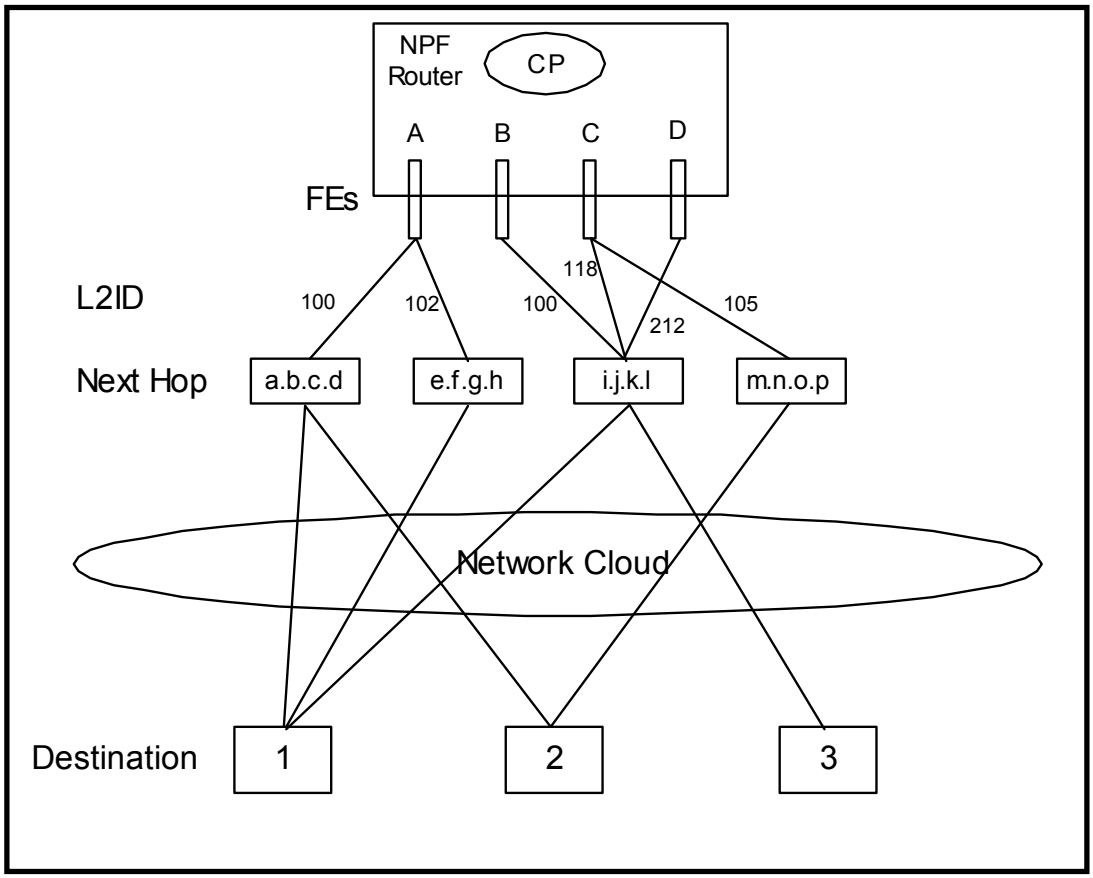


Figure 11 Hypothetical network arrangements

NPF router in Figure contains four forwarding elements, each with multiple network interfaces. Each interface on the FEs connects to one of four next-hop forwarding devices. Three destination devices exist across a network cloud. The destination device number maps to an NHID. This arrangement produces the next-hop table shown in the following table.

Note: All the possible entries are not shown in the table.

Table 16: FE Next-Hop Table

NHID	NHAddr	L2ID	FEID
1	a.b.c.d	100	A
	e.f.g.h	102	A
	i.j.k.l	100	B
2	a.b.c.d	100	A
	m.n.o.p	105	C

3	i.j.k.l	212	D
	i.j.k.l	118	C

In order to support Equal Cost Multi-Path(ECMP) algorithms in future versions of this module, multiple next-hop addresses can be associated with a NHID. Since L2ID is unique to a specific FEID, they can be identical for a given NHID. This can be seen in Table 16 for NHID 1 where an L2ID of 100 is used twice, each time for a different FEID. On FEID A, L2ID 100 maps to next-hop address a.b.c.d and on FEID B L2ID 100 maps to next-hop address i.j.k.l.

L2ID management module maintains all relationships and mappings represented in Table 16. On any given FE, an L2ID can be mapped to an L2 next-hop address without any correlation to a NHID. L2ID manager also maintains this information to make sure that each L2ID is unique within a given FE.

3.7 Modularity

Since no other components depend on IPv4, it is easy to either replace the implementation or stub out the implementation completely. As long as NPF API behavior and contracts are adhered to, internal implementation is not really important to external applications that can use IPv4 component. It is also possible to not compile IPv4 component and leave it out of PDK entirely.

3.8 Design for Multiple FE Support

This section gives basic details about the support for multiple FEs. For further details, please refer to IPv4 Manager in Forwarding Plane Module – Design Reference [1].

After getting details about FEs such as FEID that are associated with the target table, the IPv4 FP request handler module sends separate forwarding plane API to individual FE through FP plug-in API that goes down to IPv4 module in FE using the transport plug-in framework. This FP Transaction Handler module handles the logic of collating all the responses from the different FEs. Please refer to Section 3.2.2 for more details.

Logic for handling multiple FEs is handled by `IPV4TransInfo_t` member by the name `IPV4TransFPInfo_t`. Refer to the description of various fields within this structure in Section 3.3.2.

At the start of sending one or multiple requests to one or multiple FEs for a single control plane request received from NPF user application, use `numFeToSend == numFEWait`. Whenever a callback is received from a particular FE, value of `numFEWait` is decremented till it reaches zero. Once it reaches zero, a consolidated error or success return value is returned to control plane / NPF application.

If any one of the FEs return error status, the consolidated error report to the NPF user application is error.

Handling of array of entries in a single NPF request is handled by IPv4 module in forwarding plane residing on each FE. This design avoids sending individual requests for each entry in the array from CP to FP and this increases the performance and error probability. Please refer to

Forwarding Plane Module – Design Reference [1] for more details of the design for handling array of entries in forwarding plane IPv4 manager.

Part 4: NPF API to FPP API Mapping

4 NPF API to FPP API Mapping

Figure 12 illustrates mapping of the NPF APIs to forwarding plane plug-in APIs.

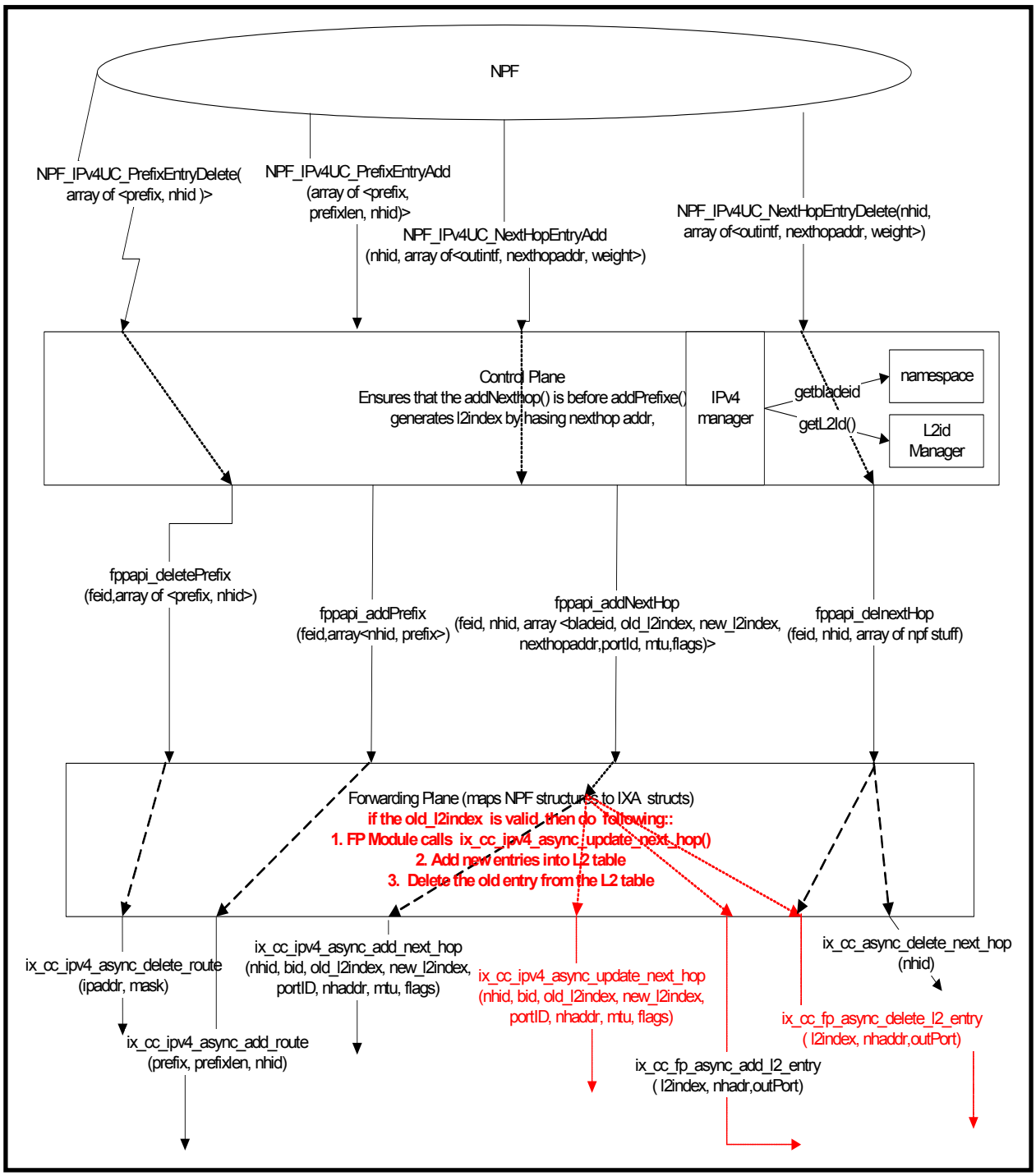


Figure 12 NPF API to FPP API mapping

Forwarding plane API expects an FE Id for each call to the FP plug-in, that is used to identify the FE to that the call is directed. The add/delete next-hop calls in the FP plug-in, additionally expect the L2 Id. FP is unable to handle adding prefixes without the corresponding next-hop. The application recommends adding prefixes before adding the corresponding next hop.

For more details on this mapping, refer forwarding plane module – design reference [1].