Software API Framework Implementation Agreement

Revision 1.0

Editor(s):

David M. Putzolu, Intel Corporation, david.putzolu@intel.com

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

Table of Contents

1 Revision History

Revision	Date	Reason for Changes
1.0	09/13/2002	Created Rev 1.0 of the implementation agreement by taking the Software API Software Framework (npf2001.042.20) and making minor editorial corrections.

2 Introduction

System vendors and independent software vendors want common application programming interfaces to allow interoperability between and among network processing devices and their associated software components. Network processing element vendors want a variety of software solutions available to expand the market for network processing. This document identifies the architectural elements of network processing solutions and ties them together in a framework for software APIs for exposing their functionality. The framework defined here enables the broad spectrum of capabilities present in network processing that exists today and will likely exist in the future. This framework is intended as the basis for using network processing elements in the construction of networking equipment.

Network processing elements are complex devices used for a wide variety of packet processing tasks. Network processing solutions use various techniques such as multithreading, multiple processors per chip, programmable state machines, high-speed interconnections, non-blocking switch fabrics, content-addressable memory, and embedded RAM to meet the increasing demands of high-end networking devices. Many network processing elements are programmable and very flexible, requiring significant investment in microcode and software development to enable higher-level protocols and applications. In some cases, network processing elements are bundled with framers for various layer one and layer two technologies. Network processing element vendors provide a wide variety of solutions satisfying the requirements of many different networking domains.

The outputs of the NPF SwAPI WG have the objective of reducing the adoption costs of using NPUs. This objective will be achieved by specifying a framework containing the key functional elements of a network device. These functional elements are defined by APIs whose operations and parameters include typical applications as well as capabilities of low-level functions.

The enterprises that will benefit from these specifications include:

- System vendors who want common service APIs to allow interoperability between and among network processing devices and their associated software components;
- Independent system integrators who can take advantage of interoperability to combine solutions using service APIs and functional APIs, general and specialized, from independent vendors;
- Network processing element vendors who want a variety of software solutions available to expand the market for network processing; and
- Stack implementers, who provide the protocols and distributed algorithms that are used to build the management and control applications that configure and monitor network elements that support the APIs.

The benefits are achieved through the level playing field that such APIs create, solving the problems of maintaining multiple proprietary code module/libraries (for suppliers and users), opening opportunities for new entrants and so on.

This document identifies the aforementioned framework, comprising architectural elements of network processing solutions, and ties them together in a framework for software APIs for exposing their functionality. The framework enables the broad spectrum of capabilities present in network processing that exists today and will likely exist in the future. It is intended as the basis for using network processing elements in the construction of networking equipment.

2.1 Framework Model Overview

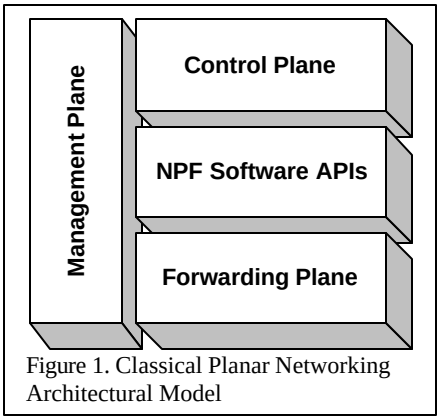

Figure 1. Classical Planar Networking Architectural Model

In order to define a set of APIs that apply to the broad spectrum of network processing offerings it is necessary to have a basic model of networking device functionality. Architectural models of common network elements typically consist of three planes – a control plane, a management plane, and a forwarding plane, as shown in figure 1. Typically, the forwarding plane consists of hardware and associated microcode or other high performance software that performs per-packet operations at line rates. The control plane, in contrast, typically executes on a general-purpose processor, is often written in a high level language such as C or C++ and is capable of modifying the behavior of forwarding plane operations. The management plane, which provides an administrative interface into the overall system, consists of both software executing on a general-purpose processor (including functionality such as a SNMP daemon and a web server) as well as probes and counters in the hardware and microcode.

The NPF architecture model takes the classical planar model and simplifies it in a functional manner. In particular, two planes of operation are identified in the NPF model: the control plane and forwarding plane. Management plane functionality is not given specific attention in the NPF model because it is closely integrated with the control and forwarding planes and is dependent upon implementation strategy. Given their roles, most management operations naturally fall into one of the other two NPF planes (e.g.: SNMP MIB population or web server execution vs. packet counting actions).

The NPF architecture, shown in figure 3 below, identifies two layers of functionality. The first layer is the System Abstraction Layer, which exposes vendor independent system level functionality. The APIs exposed at this layer, termed NPF Service APIs, are unaware of the existence of multiple forwarding planes and provide an abstraction of the underlying system that presents the functionality being manipulated by the control plane without regard to the physical topology of the system. The second layer is the Element Abstraction Layer which exposes vendor independent forwarding element aware functionality. The APIs exposed at this layer, termed NPF Functional APIs, are presumed to know which forwarding element they are addressing; this is an important assumption, as not all forwarding elements will have the same forwarding capabilities. The NPF Service APIs and NPF Functional APIs are collections of separate, related APIs.

The NPF Service APIs expose controls for service specific functionality of a device, hiding both the implementation details as well as the system composition. More specifically, the NPF Service APIs provide the illusion that they are running on a device that consists of a set of ports (of varying types) interconnected by a single switching fabric. The NPF Service APIs support such functionality as IP routing, MPLS, etc. and a set of additional, well-documented services that are exposed to the application. The NPF Service APIs are used by protocol stack and other software vendors that do not require access to implementation details and prefer a “black box” view of the packet processing elements of a system – see figure 2 as an example of this.

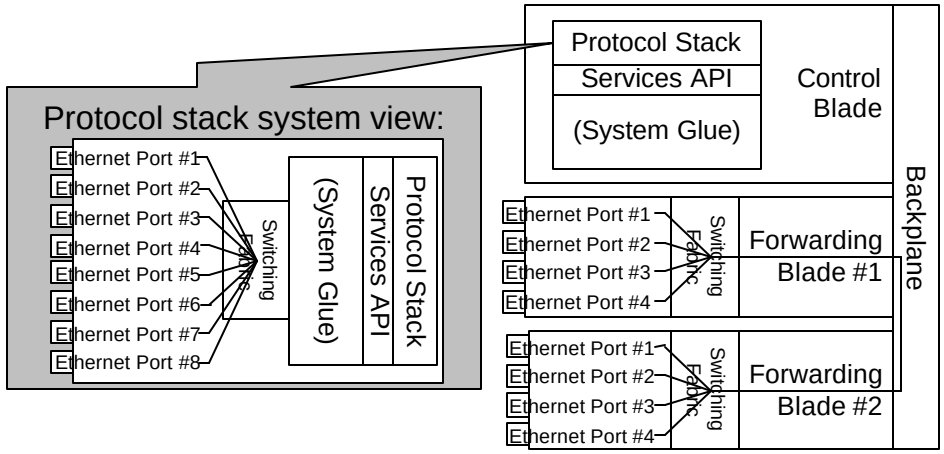

Figure 2. NPF Services API Client System View

The NPF Functional APIs hide less of the details of the system, exposing the presence of multiple forwarding devices and their individual capabilities. The NPF Functional APIs share a quality with the NPF Services APIs by the fact that they hide the vendor specific details of the network processing devices. More specifically, the NPF Functional APIs are vendor neutral, and a program using the NPF Functional APIs should be able to execute correctly on any conforming vendor's platform as long as the needed set of logical functions are present. These logical functions are interchangeably termed, "logical function blocks", "functional blocks", and "stages" and are used to represent the functionality and ordering of packet processing in the data plane. It is noted that there is a variety of functional blocks, with

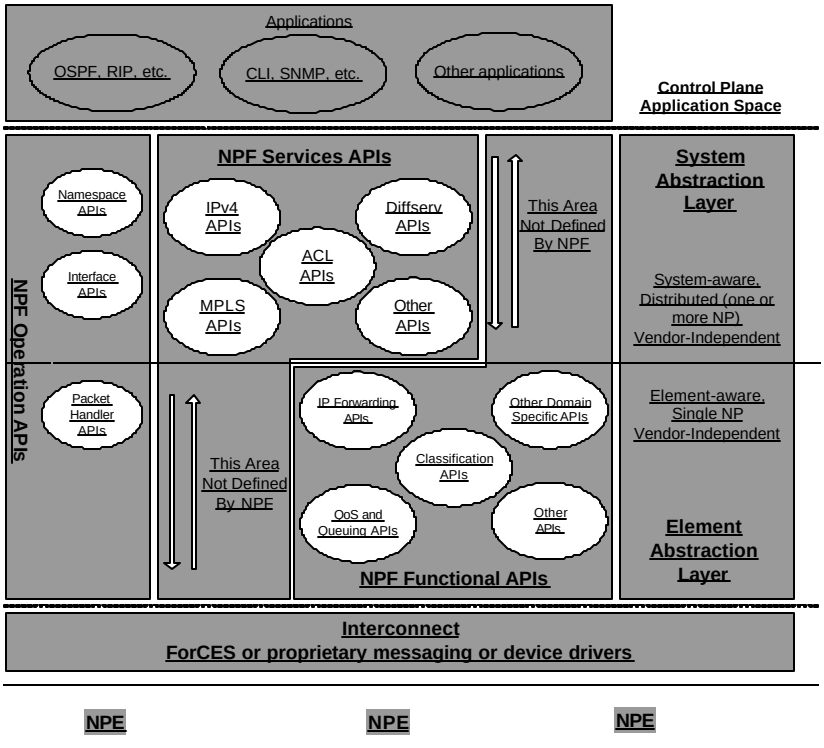

Figure 3. Overall NPF Software Model

varying capabilities, and client applications will often need to discover the capabilities of a system on which it is executing.

Figure 3 shows the overall NPF Software model. The NPF Services APIs and NPF Functional APIs make up two different layers for controlling the same basic forwarding plane functionality of a network device. The NPF Services API implementation is responsible for hiding any functional differences between the various forwarding plane solutions.

This abstraction of functionality is provided to the applications executing above the NPF Services API, either through proper configuration of the forwarding devices via the NPF Functional API or via software emulation of missing features. The NPF Functional API exposes a vendor independent model, described in section 5.3 of this document, of the functionality provided by forwarding plane devices. While the model is vendor neutral, it does expose the specific functionality available (or unavailable) from the underlying forwarding plane elements.

Both of these API sets and their client software execute on the control and management planes of a network element. The control and management planes are connected to the forwarding plane via a physical or logical interconnect.

2.2 Motivating Examples

One physical realization of the NPF model would be a Compact PCI bus between a general purpose CPU blade executing the control plane software along with line cards with classification ASICs, network processors, and framer hardware performing the actual per-packet operations. Another physical realization might have the NPF APIs executing on a general purpose CPU with directly attached framer. A third realization might have a general purpose CPU in a rack mount device executing the NPF APIs with a set of remote forwarding devices, connected via gigabit Ethernet. In the cases where the NPUs and ASICs/framer are not directly attached, it will be necessary to use a messaging mechanism of some sort. Standards such as the IETF ForCES mechanism will be used where possible and appropriate for such messaging purposes.

It is likely that some applications may want System Abstraction level control of features that are managed at a device-by-device level. Applications of this nature might prefer to specify rules that apply to parts of the system, or the system as a whole, without having to become intimately aware of the support, features, and architecture. An example application is a firewall: it would likely apply relevant classification and other actions before executing routing decisions. An alternate configuration might apply classification after routing decisions but prior to final emission of the frame. Because of this, there may be instances of similar functionality within the Services APIs and the Functional API, each presenting different levels of system abstraction.

The above is not intended to be an exhaustive list of physical realizations of the NPF Software Framework.

2.3 Document Organization

This document is organized as follows: Section 2 describes the basic architectural elements of the Network Processing Forum Software Framework. Section 3 describes the set of supported services and philosophy of the Network Processing Forum Services API. Section 4 describes the Network Processing Forum Operational APIs. Section 5 describes the Network Processing Forum Functional APIs. Section 6 explains where device-specific APIs fit into the overall software framework. Section 7 concludes the document with a summary of the overall framework.

3 Architectural Elements

Network Processing elements are often used to construct devices, such as IP Gateways (Routers), LAN bridges, and packet switches, that forward packets. They may also be used to construct more advanced devices such as web caches, intrusion detection systems, and firewalls. The overall architecture of the NPF Framework must be useful to all such categories of devices. This framework focuses on the components typically found in such devices.

We organize network node functions into three major divisions:

- The Management Plane supports the configuration and provisioning of the network node by operators and administrators. It permits monitoring of network operation, the gathering and distribution of statistics related to control plane and forwarding plane operation. It may contain operator interfaces (such as a Command Line Interface or Graphical User Interface) and management protocol implementations (e.g. SNMP) to support these capabilities. The Services API contains functions that support these management functions, whereas the Functional API provides the counters and event monitoring functions needed to implement them.
- The Control Plane maintains information for the purpose of effecting changes to the forwarding plane's behavior. In the case of a router, the Control Plane creates and maintains an accurate routing information base (RIB) that contains all known network layer connectivity information. The control plane uses the RIB to map IP destination addresses to next hops, QoS information, and other data needed by the Forwarding Plane to generate a subset that is downloaded to the forwarding plane that is called the forwarding information base (FIB) as described in RFC1812 [1]. It constantly adapts to changes in the status of its own interfaces and changes in routing and network topology information as seen through routing protocol exchanges with its peers. In the NPF Framework, the RIB is a construct of a potential routing application. The FIB contains information that would be provided via the Services API. An implementation of the Services API would then map the FIB to the various NPEs of the system according to local policy.
- The Forwarding Plane receives protocol data units on input interfaces, processes them (classification, direction, modification and traffic management), and dispatches them either to a local host processor or to output interfaces. It may perform other functions as well, such as generating ICMP error messages to the sources of packets it sees.

3.1 Management Plane

The Management Plane contains processes that support operational, administration, management and configuration/provisioning of a system. Examples of functions executing in the management plane may include:

- Facilities for supporting statistics collection and aggregation
- Support for the implementation of management protocols
- Command Line or Graphical User Configuration Interfaces

3.2 Control Plane

The Control Plane of a node maintains information that can be used to effect changes to the Forwarding Plane ensuring that, whenever possible, the forwarding plane can receive protocol data units and forward them to the next destination along a route, chosen according to some metric or policy. One commonly used criterion is that packets should take the best available path from source to destination; this requires control plane software and protocols to continuously evaluate network topology and calculate the best paths to all known destinations, recalculating whenever the topology changes. Another criterion recognizes that the payloads of some packets are more important than others, and these must be forwarded with high priority and/or only along network paths that support certain bandwidth or priority requirements (Class of Service, broadly speaking). To achieve this, control plane software must be aware

not only of network topology, but of certain qualities of the operating links. It must also maintain information that defines the means by which the forwarding plane processes packets. Sometimes forwarding is done using explicit routes, instead of the shortest paths; this usually requires modifying the state of the forwarding plane of nodes along the path of a desired route through the network. Most of the information above is maintained by software executing on top of the Network Processing Forum Services API. Such software passes the results of calculations based on the above information across the Services API to implement the proper packet processing treatment by NPEs.

3.2.1 Protocols

In general, the Control Plane does its work using Protocols that operate between nodes, either directly connected or at some distance over a network. In the case of IP packet processing, there are four broad categories of protocols:

- Routing protocols – OSPF, IS-IS, RIP, BGP, MOSPF, DVMRP, PIM, BGMP, etc.
- Signaling protocols – RSVP, RSVP-TE, LDP, Q.2931, etc.
- Discovery protocols – ARP, ICMP, IGMP, etc.
- Informative protocols – ICMP

3.2.2 Packet I/O for Applications

The Packet Handler APIs provide a path for applications to send and receive packets through Forwarding Elements (FEs). The Packet Handler comprises a service level API that supports input and output sessions for individual applications, and a functional API that communicates directly with a Forwarding Element (FE). The following diagram shows the relationship of the two APIs and their applications.

Clients of the service level Packet Handler API include network processing applications (some of these may logically reside in the forwarding plane), socket "device" drivers that support the TCP/IP stack in the operating system's kernel, and user level applications that need "raw" access to NP devices to send and receive packets.

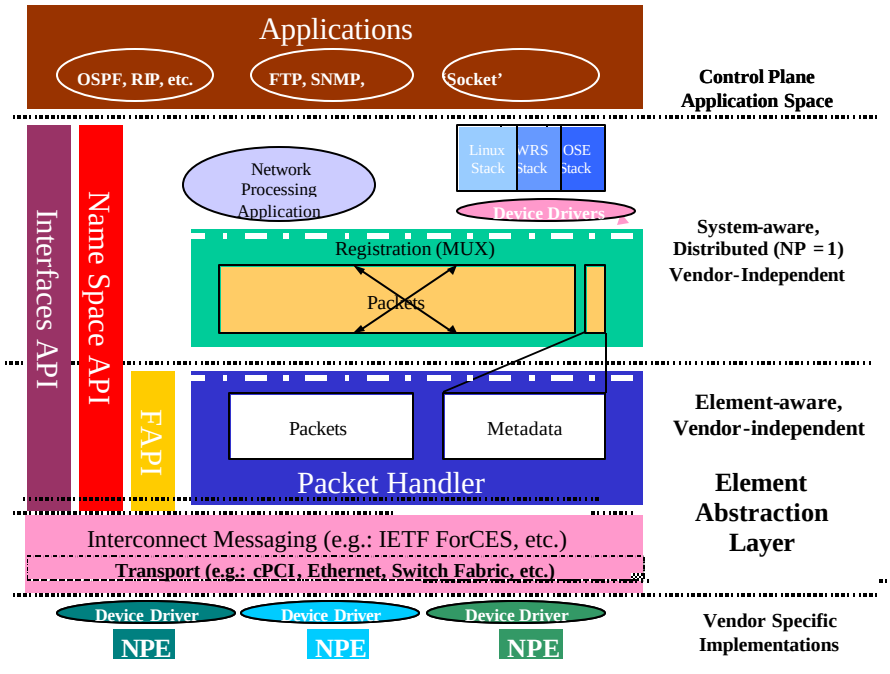

Figure 4. Packet Handler Architecture

System-aware code beneath the service level Packet Handler API (the green box in the above diagram) demultiplexes and dispatches packets between these clients and the functional Packet Handler APIs to individual NP devices.

3.3 Forwarding Plane

The forwarding plane is responsible for performing wire speed data forwarding operations and emitting exception packets to the control plane. The types of forwarding operations that are available are dependent on the interfaces that exist on the forwarding plane and protocols used to make forwarding decisions. In this section we'll examine the sorts of interfaces and protocols that will need configuration in the forwarding plane.

3.3.1 Interfaces

Network processing elements send and receive network datagrams on Interfaces. The word interfaces is used in many ways, sometimes referring to specific hardware ports, sometimes "virtual" hardware, and so on. Network processing elements concern themselves with interfaces of many types; but the central idea of an interface, for network processing elements, is a logical construct that binds packets of particular protocols to particular physical or logical ports. The physical interfaces that participate in this binding can take many forms: physical ports, virtual connections (ATM, for instance), or even multiples of those. Interfaces can be either unidirectional or bi-directional, but when unidirectional interfaces are used, they are often configured in pairs to permit traffic to flow in both directions between two peer systems. An MPLS path is one example of a unidirectional interface.

For any given protocol, whatever the physical construct, a logical interface, as defined here, connects a network node to exactly one network, to which are attached one or more peer network nodes that are one hop away –that is, packets travel between nodes using link-level forwarding only. Peers attached to a network have a common view of the network and what protocols are valid on it. Lower-level protocols such as Point-to-Point Protocol (PPP) can be used between a pair of peers to negotiate the commonly accepted set of higher level protocols (e.g. IPv4, IPv6, MPLS, and so on) that will operate on an interface.

Interfaces typically have a set of attributes that do not depend on the choice of network protocols they carry, but are related to the underlying transmission links. These attributes include a maximum data rate or bandwidth, a link-level address, an agreed-upon set of protocols to frame datagrams. The interfaces carry link-level address information, a maximum permitted frame size, information as to the operational status (up or down) of the links, and some identification of the systems at each end-point that is directly reachable via an interface.

Interfaces also have assignable attributes related to the protocols they carry. For instance, IPv4 uses maximum transmission unit (MTU), an administratively-enabled flag, and many other attributes related to routing protocols that may be of interest to the Network processing element.

Most of these attributes are allowed to change during normal operation without requiring any interruption of forwarding or traffic flow. A great many provisionable objects in a node are present in the interface table, so that they can be scoped to an individual interface. For instance, enabling a protocol such as IPv4, OSPF, etc., is typically done on one interface at a time rather than for the system globally.

3.3.2 Forwarding plane architecture

The forwarding plane is a subsystem of a network node that receives packets from a framer on an interface, processes them in some way required by the applicable protocol(s), and delivers, drops, or forwards them as appropriate. The essence of a Forwarding plane element is that it offloads packet forwarding chores from "higher-level" processors: for most or all of the packets it receives that are not addressed for delivery to the node itself, it performs all required processing. The functional blocks involved in protocol data unit processing can be classification, direction, modification (including encryption/decryption) and traffic shaping.

Some forwarding plane implementations include interfaces to optional devices that assist with special operations that are ordinarily time-consuming or computationally intensive, so that packets needing these processing steps can still be forwarded at high speed. Forwarding plane architectures that are distributed over a set of processors and specialized coprocessors are as justifiable as architectures that involve a single processing element. Such distributed operations might include checksum generation, classification, encryption, packet modification, traffic management, etc.

3.3.3 Protocol Data Unit Processing

Protocol Data Unit processing embraces all the means by which a network-processing element determines how & where a packet should be forwarded, what preferential treatment it should be given, if any, and how it should be modified in the process.

3.3.3.1 IP Processing

At least in the abstract, IP processing uses a number of specific data structures. Specific references to tables and other structures below are only meant to be exemplary rather than refer to Network Processing Forum API implementations or data types. IP processing includes, but is not limited to:

- The Interface Table – a list of interfaces belonging to a common domain or address family. This table should indicate which protocol types are valid and should be forwarded on a given input link, what is the largest protocol data unit that can be forwarded without fragmentation on a given output link, and so on.
- The Forwarding Information Base (FIB)
- The Diffserv Map – a 64-entry table that gives QoS or CoS specifications for every possible Diffserv Code Point (DSCP).
- The Filter Database – a list of packet templates against which incoming packets are compared. The templates will indicate the required disposition for matching packets, for instance unconditionally count and drop. Network administrators often create filters to rid their networks of undesirable traffic.
- The MPLS Label Map – a lookup table used in forwarding MPLS encapsulated packets; it maps an incoming label value to an outbound link and a new label value.

3.3.3.2 Link Level Switching

Link Layer switching applies to several media types; Ethernet and ATM are common examples, while HIPPI is a less common example. Provisioning of the forwarding information in these networks can be provided either statically or signaled via out-of-band protocols. The following two sections describe the interfaces needed by specific instances of Link-Level forwarding technologies.

3.3.3.2.1 LAN Bridging (Ethernet Switching) Processing

This section describes data types, structures, APIs, and manifest constants that are used to specify operation of a device as a LAN bridge. LAN bridging processing includes, but is not limited to, operation as described by the 802.1 sets of standards.

In the abstract, this processing includes:

- Adding, modifying, and removing layer 2 filtering database entries (i.e. MAC addresses)
- Configuration of the operational or Spanning Tree state of a LAN port or network interface
- Definition of the set of filtering databases to be held in the network processor, and the parameters under which they operate (i.e. aging times for dynamic entries and the VLAN IDs associated with the filtering databases)
- VLAN configuration of LAN ports (ingress rules, egress rules, and VLAN membership rules)

3.3.3.2.2 ATM Switching

ATM switching information includes.

- Per port VPI/VCI forwarding databases.

4 NPF Services APIs

These APIs support the creation and maintenance of tables and processing rules that govern the processing of forwarded traffic. These service-specific APIs provide a high level of abstraction and are aimed at facilitating the implementation of networking software executing over hardware platforms with a high degree of functional integration. Each API family includes targeted tools for its configuration and maintenance including managing protocol-specific statistics collection and aggregation.

The examples in the following sections do not represent a complete list of APIs or necessarily reflect the order that task groups will be chartered to address these APIs.

4.1 IPv4 API

The IPv4 APIs in the Services API set are used to control the IPv4 forwarding of a network element that implements the Network Processing Forum APIs. It should be noted that the IPv4 API is used to manipulate the Forwarding Information Base (FIB) of a device, and that applications executing above the API are expected to contain and manipulate the Routing Information Base (RIB), reducing it to a FIB that is then passed across the IPv4 API. In this context, the RIB is the complete set of route information from the set of routing protocols on a device, and may include conflicting routes from different sources (e.g. RIP and OSPF protocols). The FIB is a coherent collection of forwarding information that is typically generated according to the policies of the network administrator and is a subset of the RIB.

4.2 IPv6 API

The IPV6 APIs are a unique set of Services APIs that will be utilized to provide support for IPV6 forwarding for a network element that implements the Network Processing Forum APIs. The goal of this API set is to be integrated into new or existing control plane applications that will update the FIB with current routing information provided by optimization of routing information from the RIB (Routing Information Base). These APIs will provide a mechanism that will allow for a 128 bit IPV6 address which supports more levels of addressing hierarchy and a much greater number of addressable nodes.

The functionality required of the IPv6 API set is similar to that described above for the IPv4 APIs.

4.3 DiffServ API

Differentiated Services is an architecture for providing different types or levels of service for network traffic. The differentiated services approach to providing QoS in networks employs a small, well-defined set of building blocks from which a variety of aggregate behaviors may be built. One key characteristic of DiffServ is that flows are aggregated in the network, so that core routers only need to distinguish a comparably small number of aggregated flows, even if those flows contain thousands or millions of individual flows. A replacement header field, called the DS field, is defined by Differentiated Services. The DS field supersedes the existing definitions of the IPv4 type of service (ToS) octet (RFC 791 [2]) and the IPv6 traffic class octet. Six bits of the DS field are used as the DSCP that is used to determine a particular forwarding treatment, or Per Hop Behavior (PHB) that the packet will receive at a network node.

The DiffServ API provides an interface for setting up various DiffServ building blocks so that a packet will receive the correct PHB at a node.

4.3.1 DiffServ Statistics API

This API supports the definition, creation and maintenance of DiffServ counters and other traffic statistics within the forwarding devices.

4.4 MPLS API

Multiprotocol Label Switching as defined in RFC3031 [2] is an IETF standard that integrates the control of IP routing with the simplicity of Layer 2 switching. As with any multilayer switching technique, MPLS is composed of two distinct components: control component and forwarding component. The control component uses standard routing and signaling protocols like OSPF, BGP, RSVP, and LDP, to exchange information with other routers to build and maintain the forwarding tables. MPLS control protocols (CR-LDP, RSVP-TE, OSPF-TE, etc.) are beyond the scope of the Network Processing Forum Framework and are implemented as applications executing on top of the MPLS Service APIs.

The MPLS forwarding component is based on the label-swapping algorithm similar to ATM and Frame Relay switching. A label is a short, fixed-length value carried in the packet's header to identify a label-switched path (LSP) that the packet must follow through the network.

The Network Processing Forum MPLS Service API provides interfaces for controlling the switching and addition/removal of labels and label stacks and the QoS treatment of MPLS framed packets as well as controlling the admission and egress of packets from MPLS domains.

4.4.1 MPLS Statistics API

This API supports the definition, creation and maintenance of MPLS counters and other traffic statistics within the forwarding devices.

4.5 Access Control List API

The Access Control List (ACL) management API enables an application to manage a set of access control list entries. This API offers service support in order to enable permit and drop actions to be processed by the network processing elements. The ACL management API enables implementation and activation of the ACL entries in order to facilitate packet bit stream identification.

4.5.1 ACL Statistics API

This API supports the definition, creation and maintenance of ACL counters and other traffic statistics within the forwarding devices.

5 NPF Operational APIs

The Network Processing Forum Operational APIs are used for the control and management of functions that are present both at the system abstraction layer and at the element abstraction layer. Network Processing Forum Operational APIs include interface management and statistics, layer 2 address assignment, packet insertion and extraction APIs, and system namespace APIs.

5.1 Interface Management API

This API provides a set of functions used to manage Network Processor interfaces on which frames are sent and received.

5.1.1 Interface Statistics API

Counters managed by this API are associated with interfaces. The Interface Statistics API facilitates querying the device for the counter values defined below. Each counter is uniquely identified according to its type and name.

Interfaces have several associated attributes. These attributes include counters (such as frames sent, bytes received, various errors), states (such as link up, speed, duplex), and general information (such as type). Additionally, some interface types have additional data applicable to several types of interfaces (such as address) or other data that only applies to a specific interface type (such as ATM QoS specifications). These attributes, along with how they are manipulated (such as enumeration, reading, writing), are described in the interface management API specification.

5.2 Packet Handler API

This API permits applications to send and receive packets through the forwarding devices. It supports origination and termination of traffic by protocol implementations and other application-level software. Packets traversing the system are processed by the network processing element or they are processed by the control plane software running on top of the Services APIs. The Packet Handler API is used to deliver and accept frames from the control plane software.

6 NPF Functional APIs

The NPF Functional API provides support to an application in the terms that are function specific. The NPF Functional API implements the support for network processing functions in a manner that is targeted to directly access the functionality offered by network processing elements.

6.1 Introduction

The NPF Functional API is expressed using a model of forwarding plane functionality. The NPF Functional model is expected to be compatible with the ForCES Framework and Model that are being defined by the IETF ForCES working group as shown in figure 3 above. This model is used to describe the forwarding plane's functionality. The control plane may dynamically discover the functions of a forwarding plane as expressed via the functional model, or a static description of a forwarding plane may be provided that is used during system integration to custom-design a control plane. Regardless of whether discovery occurs, the control plane uses the NPF Functional model to manipulate the functionality of the forwarding plane.

6.2 Usage

The Network Processing Forum Functional API may be used by implementations of Services APIs and other applications.

The Network Processing Forum Functional API must provide all the facilities needed to efficiently implement all the Network Processing Forum Services APIs supported by the forwarding plane functionality. Services API implementations may bypass the Network Processing Forum Functional API.

6.3 Functional Model

The NPF Functional model (or "model" for short) of the forwarding plane represents the functionality of the forwarding plane as a sequence of functional blocks termed "stages". These stages represent the set of functions supported by a data plane, along with their ordering. The functional blocks may be arranged in a graph, with different stages applying to different subsets of packets. In such configurations a stage may have multiple stages to which its outputs are "connected" delivering packets to a different downstream stage based on its own internal classification of packets. These stages and the sequence they are organized in is a purely logical representation of data plane processing and is not tied to actual data plane implementation.

Conceptually, each stage performs some well-specified logical function. Given that new types of network functionality evolve over time, the stage model must reflect the existing state of the art and allow extensions to be specified to include new functions. Stages include one or more capabilities and operate on either the packet data itself or some metadata associated with each packet – e.g. a flow ID. The capabilities define the rules it applies to select packets, the actions it performs, parameters it uses for those actions and its basic logical function, the events it may generate, and the statistics it collects.

The FAPI does not prescribe the set of stages required to implement the services performed by a FE. A vendor of a FE supporting FAPI must support the minimum set of functional APIs to allow stage capabilities to be configured but, in the FE implementation itself, Stages may represent anything from very simple functionality (e.g. a dropper stage which provides a counter of dropped packets) to very complex functions (e.g. an IP forwarder, a security stage capable of encapsulating packets in IPSec using tunnel or transport mode). It is an implementation decision, for example, to model an octal eight port egress stage as eight individually named egress stages or as a single stage with eight separate property capability sets.

An example of an IPv4 routing blade is given in section 6.5 below to illustrate this. If the device also implemented DiffServ functionality then additional stages would be needed to meter, mark, drop and queue packets.

6.4 Functional API

The Functional API is based on the model described in section 6.3 and consists of two parts. The first part optionally provides a set of methods for discovery of the stages and capabilities supported by a forwarding device. The second part provides methods for the manipulation of the capabilities of each stage in the forwarding device.

6.5 IPv4 Forwarding Blade Functional API Examples

This section describes how the Functional API and its model could be used to represent and control simple IPv4 forwarding blade functionality. Two different potential implementations of forwarding blades are described. In both cases, the same Functional API methods are used, however, they are used in different combinations for each example.

6.5.1 Exact-Match IPv4 Forwarding with Static Configuration

In this example, a blade is being integrated into a system using the functional API. The forwarding blade functionality as well as its representation via the FAPI is known at system integration time, perhaps via vendor datasheets. The datasheets indicate that the forwarding blade is modeled as three stages – ingress, IPv4 forwarding, and egress. The ingress stage represents eight fast Ethernet ports, the IPv4 forwarder stage performs exact match forwarding and requires a next hop MAC address, and that the egress stage represents eight fast Ethernet ports. Furthermore, the IPv4 forwarder stage exposes events that represent IPv4 forwarding table misses.

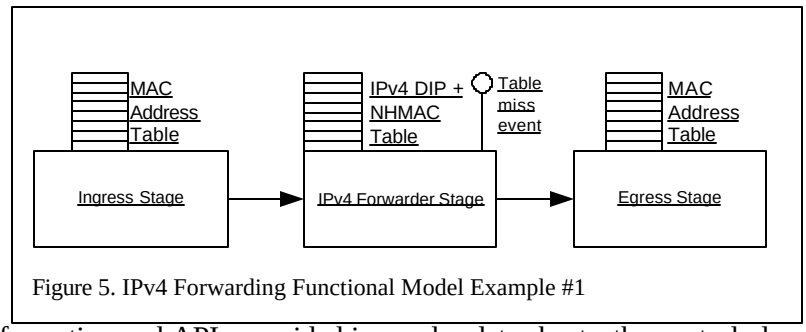

Figure 5. IPv4 Forwarding Functional Model Example #1

Using the stage information and APIs provided in vendor data sheets, the control plane configures the three stages of the forwarding blade. Such configuration information might include MAC addresses and a set of forwarding entries, as well as registration for table miss events. This actual APIs used would consist of invocations of the table manipulation, event registration, and parameter modification methods found in the Functional API.

The IPv4 forwarder represents a stage that classifies packets and creates metadata that informs the egress stage which port to use for its next hop. Note that the ARP functions that build the mapping between MAC addresses and the FIB for ingress and egress are not shown in this model. ARP can be implemented

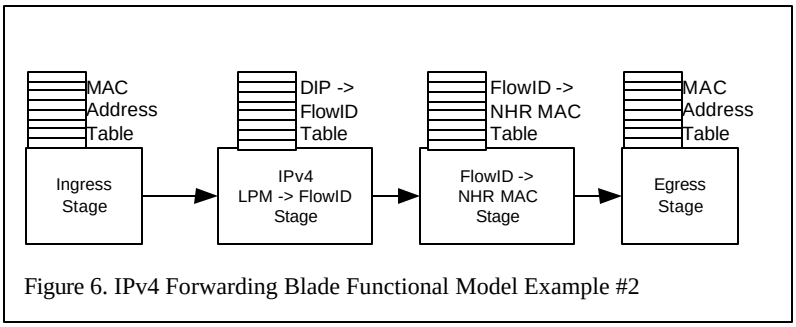

Figure 6. IPv4 Forwarding Blade Functional Model Example #2

in several ways, e.g. by generating a MAC table miss event in the egress stage, causing the CE to generate a resolution request via packet-handling APIs to discover the IP/MAC relationship. Alternatively, ARP could be implemented entirely in the blade with local stage feedback if needed.

6.5.2 Longest Prefix Match IPv4 Forwarding with Discovery

In this example, a forwarding blade is integrated into a system at runtime, and a set of discovery methods are used to discover the set of stages used by the forwarding blade to model its functionality. When the forwarding blade is initially connected to a control plane (perhaps at system startup), the control plane must determine the type of forwarding blade and the functionality that it provides. Using the discovery APIs provided in the FAPI, the control plane discovers that the forwarding blade is represented as four stages. The ingress stage represents eight fast Ethernet ports, the IPv4 forwarder stage performs longest prefix match forwarding and associates a flowid with each packet, an next hop resolution stage indexes the flowID to a next hop router and its MAC address, and an egress stage represents eight fast Ethernet ports.

7 Device Specific APIs

The Network Processing forum does not address APIs for management of forwarding devices by device-specific application software. Such operations include loading, booting, configuring, code download, stopping, diagnostic debugging, dumping, etc., of the forwarding devices. It is expected that such APIs will contain significant amounts of opaque, vendor- and platform-specific data.

8 Conclusion

The goal of the Network Processing Forum is to foster the adoption of Network Processing technology by creating an environment where system vendors can rapidly integrate network processing elements, protocol stacks and applications to deliver solutions to customers. This document has described the software model that will facilitate this goal. It is not required that all the elements of this framework be present in every platform that utilizes network processing elements. The major features of the framework that allow this rapid integration are the Services APIs and the Functional APIs.

The Services APIs enable system vendors to integrate portable protocol stack offerings and applications without concern for the underlying network processing element that will be providing the per- packet processing specified by the protocol or application. Broad adoption of the Services APIs will ensure vendors of networking devices that they will be able to choose best of breed software components and avoid re-implementing their system when choosing new software components.

The Operational APIs provide access to functionality needed at all levels of a system, including basic configuration of attributes as well as the ability to send and receive packets by a control plane application. The Operational APIs may be used in combination with the Services APIs or the Functional APIs to provide complete aspects to all attributes of a network processing system.

The Functional APIs allow a mapping of the services enabled by the Services APIs to specific network processing elements while providing a consistent interface. While it is true that the mapping may not be the same for all system vendor designed platforms, the presence of the Functional APIs guarantee that the application can be mapped.

In addition to providing an outline and example instances of the Services and Functional APIs, this document also provides a description of the architectural assumptions about the interfaces between these APIs and other subsystems of a system vendors platform. Further information concerning the specifics of the individual APIs will be developed in the Forum and will be made available as they are defined.

9 References

1 Baker, Fred (ed.), “Requirements for IP Version 4 Routers”, Internet Engineering Task Force RFC 1812, June 1995.
2 Postel, John (ed.), “Internet Protocol”, Internet Engineering Task Force RFC 791, September 1981.
3 E. Rosen, A. Viswanathan, R. Callon, “Multiprotocol Label Switching Architecture”, Internet Engineering Task Force RFC 3031, January 2001.

Appendix A. Sample Configurations

A.1 Configurations of forwarding plane elements with framers and switch fabric

Network processing elements typically find themselves in two places in a hardware architecture:

1. on the ingress, receiving packets from an input framer and passing them on to a switch fabric
2. on the egress, receiving packets from the switch fabric and sending them to an output framer.

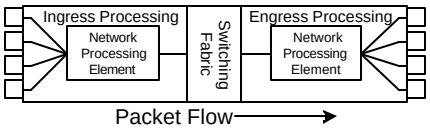

Figure 7. Network processing element hardware configurations

Figure 4 shows a configuration with network processing elements appearing both on ingress and egress of a device.

Other configurations are also possible. The simplest with respect to the forwarding device might be a system that includes just one forwarding device serving multiple input and output interfaces.

A.2 Multiple network processing element forwarding blade configurations

Another simple configuration is one with multiple network processing elements, with each one dedicated to an input or an output interface, or to an input/output interface pair. Figure 5 shows an example of this, where a NPE is dedicated to each ingress and each egress framer.

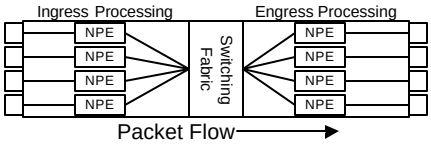

Figure 8. Network processing element per port

A.3 Multi-chassis architectures

The Network Processing Forum architecture is used in configurations consisting of multiple chassis connected together via a suitable interconnect such as gigabit Ethernet. Examples of such chassis include stackable routers and switches. Figure 6 is an example of such a configuration where there might be multiple network processing elements in use in the several of the chassis.

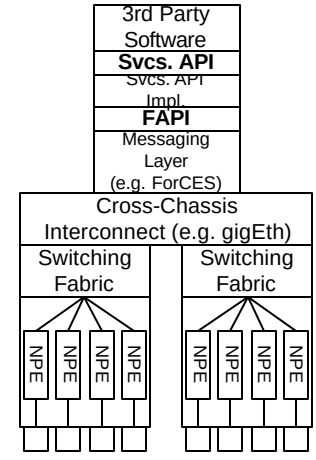

Figure 9. Example multi-chassis NPF architecture