Software API Framework Implementation Agreement

Revision 2.0

Editor(s):
Gamil Cain, Intel Corporation, gamil.cain@intel.com
Jayaram Kutty, IDT, Jayaram.Kutty@idt.com
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.
2.0	2/3/2006~~2/2/2006~~	Updated to capture high availability, Functional API model, ATM and Network Application architectural requirements, along with other general edits.

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 Software Working Group have the objective of reducing the adoption costs of using network processing elements. 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. The framework additionally enables use of these capabilities in networking elements that require a high degree of availability (sometimes termed “five 9’s” of availability), allowing NPF compliant devices to be deployed in carrier grade systems. The framework is intended as the basis for using network processing elements in the construction of networking equipment.

The framework also recognizes that network elements are increasingly performing content processing, requiring more and more compute resources. At the same time high density compute platforms are increasingly requiring more and more network processing resources. This framework eases the integration of compute and network processing in order to lower cost and increase security and performance of networked systems that integrate network elements and compute elements.

2.1 Terminology[1]

Term	Description
API	Application Programming Interface
ATM	Asynchronous Transfer Mode
CE	Control Element
FAPI	Functional API
FE	Forwarding Element
FIB	Forwarding Information Base
ForCES	Forwarding and Control Element Separation
HA	High Availability
IA	Implementation Agreement. Officially released (published) NPF specifications.
LFB	Logical Functional Block
MIB	Management Information Base
NE	Network Element
NPE	Network Processing Element
SAPI	Service API
SDI	Services Discovery API
SLB	Server Load Balancing
SNMP	Simple Network Management Protocol
SSL	Secure Socket Layer

2.2 Framework Model Overview

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

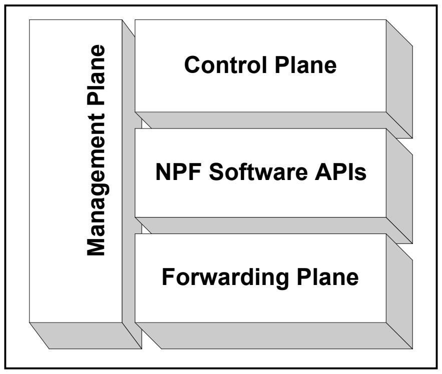

Figure 1. Classical Planar Networking Architectural Model

[1] [13] Contains additional terminology information.

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).

Certain network services provide health, environmental and application based feedback to the forwarding plane so that the network can dynamically adjust to changing conditions of the network services. The model shown in Figure 2, illustrates how the control plane takes inputs from a variety of sources. These sources include but are not limited to: Network Service Policies, Application Agents, Environmental Agents and Administrative input. Network Service Policies, for example, are the policies that forwarding plane applications such as SSL Proxy, Load Balancing, or IP forwarding may require. Application Agents provide information that dynamically adjusts the behavior of the forwarding plane application based on the feedback a networked application is providing. For example, the load of a specific application on a server can be provided to indicate how a load balancer should either steer or not steer applications to the application running on the server. Environmental Agents provide feedback on the health of various elements in the network and adjust the behavior of the forwarding plane application. The framework supports this model where multiple agents can dynamically provide inputs to the control plane and thus dynamically influence the behavior of the Forwarding Plane.

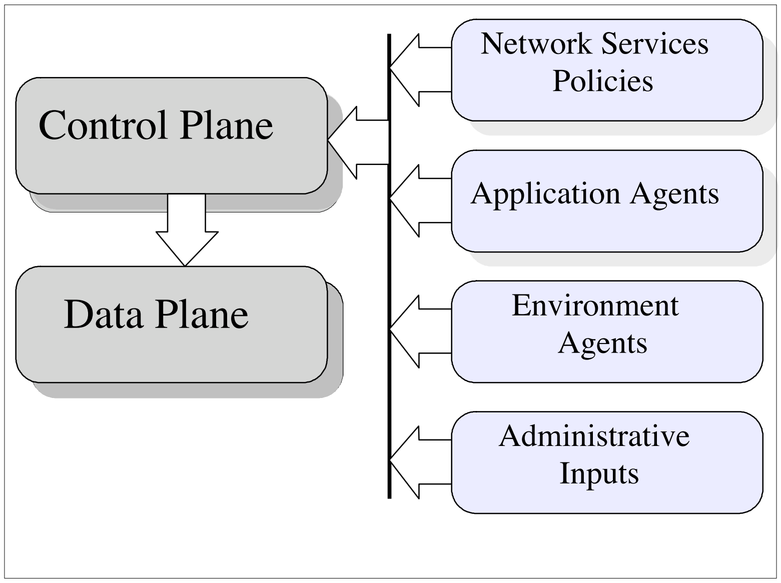

Figure 2 -- Control Plane Inputs

The NPF architecture, shown in Figure 4 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 3 as an example of this.

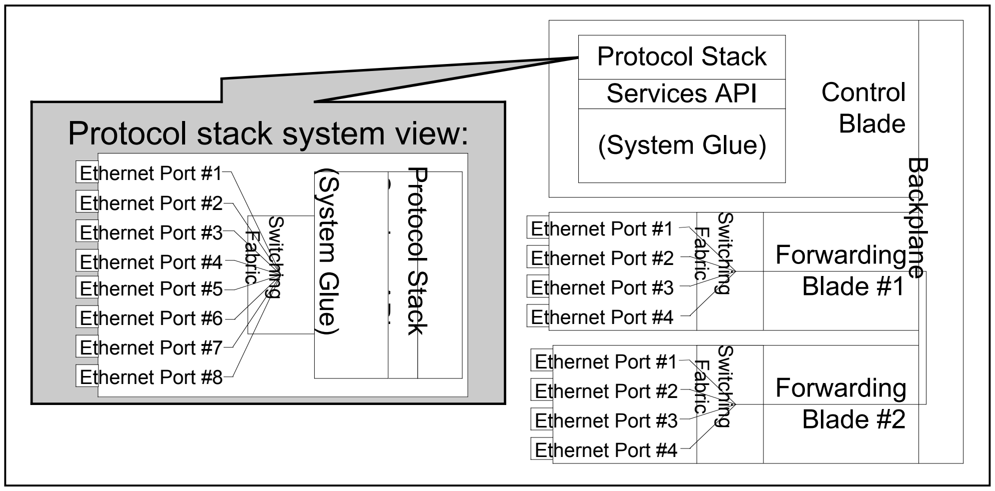

Figure 3 NPF Services API Client System View

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

It must be noted that a complete system would include a system level infrastructure whose definition is beyond the scope of the NPF APIs. Examples of such system level infrastructure are Route Discovery Algorithms, Signaling protocols, Management and Billing applications. This system level infrastructure would maintain any system level information and would interact with the NPF APIs to control the supported components of the system. While the NPF APIs expose the functionality of the system that it supports, they do not by themselves result in a complete system.

Figure 4 provides a conceptual representation of the NPF Software model. Implementations of this model may be realized on a variety of different platforms, including but not restricted to, single board computers, a multi-blade chassis, and pizza box systems. Additionally, how one chooses to implement NPF APIs within an Operating System (i.e. in kernel space, in user space, etc.) is strictly an implementation choice and not dictated by NPF.

The NPF Network Applications (NA) API are used for configuration and management of different network services such as Server Load Balancers (SLBs), SSL acceleration applications, Stateful Firewall applications, etc. The role of NPF NA APIs is to support network applications permitting the integration of multi-vendor data center processing and network service platform devices, enabled by the use of Network Processing Elements (NPEs). The motivation for defining such APIs comes from a few observable industry trends which are driving integration of advanced network services. The first is the need to provide more secure and scalable networks. The second is protection of application servers and clients. The third is the need to reduce the cost and complexity of deploying and managing these services.

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 (including NPF Network Application APIs), 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 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.

The NPF Services API and NPF Functional API support both a single client and a multiple client model.

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.

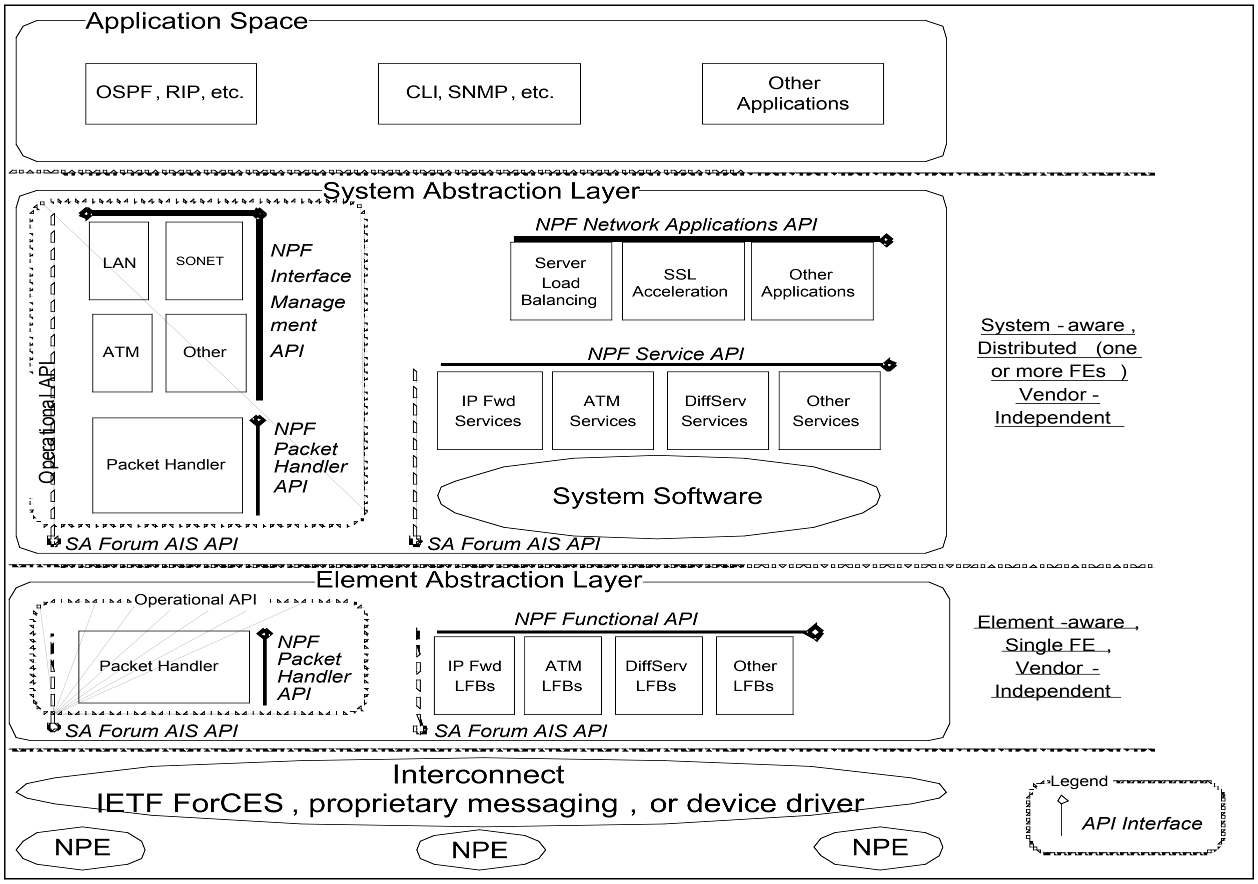

Figure 4 – Overall NPF Software Framework Model

In order to satisfy the requirements of carrier grade systems, the NPF Software Framework additionally recognizes the capabilities specified by the Service Availability Forum (SA Forum) as an effective means for integrating High Availability (HA) features in an NPF compliant system. The HA components, and their APIs, defined by SA Forum can be logically classified as operational components and APIs. NPF API implementations may make use of these HA services in order to enable replication and notification services and/or invoke resource management functions for seamless redundancy/fail-over mechanism for applications. The HA related components, within the NPF Software Framework model, does not provide direct interfaces to the NPEs and services within a network element.

2.3 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.

As there are variations in the physical realization of the NPF Software Framework, there are also variations in usage models of the APIs defined by the Framework. Figure 5 provides an illustration of the types of control paths that could exist in a network element. The figure illustrates some of the possible control paths supported by the NPF Software Framework.

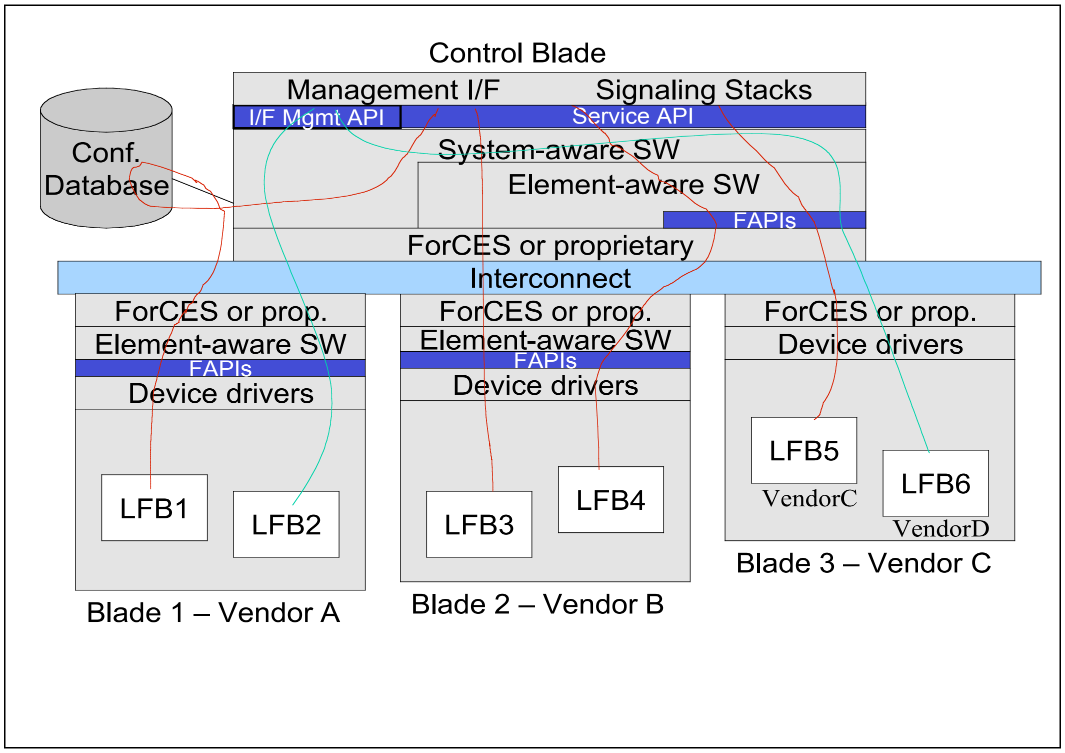

Figure 5 -- Example control paths in a network element

While the implementation of any NPF APIs is out of scope for NPF, there are some general assumptions that can be made about the types of software that may be realized in implementations.

The NPF Interface Management API, which is considered an Operational API, may reside within the control blade of a network element. It provides an interface for managing external network interfaces (ports) in the network element, either logical or physical. It further provides the ability to monitor the health of system interfaces via the collection of statistical data and events for those interfaces.

The NPF Service API (SAPI) resides within the control blade of a network element. It provides control plane applications (signaling stacks, etc.) the ability to manage a particular service (e.g. ATM). The implementation of the Service API will include some amount of system aware software and potentially some amount of element aware software. The purpose of the Service API implementation is to utilize the elements of the forwarding plane to perform the functionality of a service. This will require accessing specific Logical Functional Blocks in the forwarding plane and implies that the Service API implementation understands the physical makeup of the forwarding plane (i.e. which LFBs reside on a particular forwarding blade). Some of the actions that the Service and Interface Management API implementations may perform on these LFBs include passing configuration information or modifying the behavior of the LFB in some manner. The Service and Interface Management API implementations will manage LFBs via the NPF Functional API (FAPI).

As illustrated by Figure 5, there are variations in where the FAPI interface may reside. In some cases, the FAPI interface may reside on a forwarding plane blade (as illustrated by Blades 1 and 2 in Figure 5). In such cases, calls to the FAPI interface are made via proprietary means or by implementing a standard interconnect protocol such as IETF ForCES. In other cases, a FAPI interface may be exposed in the control blade itself. In order to transfer the FAPI function call to the forwarding plane blade containing the implementation of the function call (LFB), a proprietary or standard interconnect protocol may be used to exchange data with the LFB. The implementation of the NPF Functional API may be a mapping to a vendor specific device driver and/or may contain logic that directly controls the behavior of the LFB.

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

2.4 Document Organization

This document is organized as follows: Section 3 describes the basic architectural elements of the Network Processing Forum Software Framework. Section 4 provides a brief description of the tools the NPF Software Framework provides to aid applications in resource management. Section 5 describes the philosophy of the Network Processing Forum Services API. Section 6 describes the Network Processing Forum Operational APIs. Section 7 describes the Network Processing Forum Functional APIs. Section 8 explains where device-specific APIs fit into the overall software framework. Section 9 describes how the framework can be applied in a highly available system. Finally, section 10 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. Network Processing Elements can also be integrated with computing elements to implement advanced, application specific devices like SSL accelerators and Server Load Balancers. 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 element functions into three major divisions:

- The Management Plane supports the configuration and provisioning of the network element 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 NPF Services API contains functions that support these management functions, whereas the NPF 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. Its responsibilities include, but are not restricted to, forwarding plane configuration, execution of routing and signaling protocols, and call processing.
- The Forwarding Plane receives protocol data units on input interfaces, processes them (e.g. classification, direction, modification, traffic management, etc.), 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.

This is a general characterization of the management, control and forwarding planes. There are exceptions to this generalized view, which the NPF Software Framework also addresses. Certain advanced network services, such as SSL acceleration, may offload some or all of control plane functionality to the forward plane. The control plane would then use the Service Discovery API to discover the services being provided by the Forwarding Plane.

In carrier grade systems, interleaved within each of these planes will be support for high availability. HA services, such as state replication and fail-over, must be integrated into each plane to eliminate any single point of failure in a network element.

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

As previously mentioned, the NPF model does not give specific attention to the Management Plane since the implementation of a management sub-system is highly dependant upon the type of network element being managed. The NPF model can be viewed, however, as a complimentary piece to a management sub-system by providing some of the interfaces required to manage NPEs in a network element.

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

A variety of network elements support the origination and termination of network traffic (packets). To support these operations it is necessary to establish a conduit by which packets entering a network element can be processed by the application(s) running on the network element and similarly provide a conduit for applications to insert packets into the network. This conduit is generally termed Packet I/O. Details on how the NPF Software Framework comprehends Packet I/O can be found in Section 6.

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 interface is used in many ways, sometimes referring to a specific hardware port, 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 element to exactly one network, to which are attached one or more peer network elements 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 element that receives packets from a framing device (i.e. MAC or optical 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 Processing Protocol Data Units

Protocol Data Unit (PDU) processing embraces all the means by which a network-processing element determines how and where a packet should be forwarded, what preferential treatment it should be given, if any, and how it should be modified in the process. Appendix C provides some examples of PDU processing that can take place in the Forwarding Plane and is included to add clarity to the functions performed in the Forwarding Plane.

3.4 System Level Elements

3.4.1 High Availability

As was previously mentioned, carrier grade systems impose certain additional requirements on NPF compliant network processing elements. Network elements may have to maintain per-user or aggregate states to satisfy the service requirements of emerging real-time services. Hence, these network elements must provide high-availability features in order to avoid disruption in service.

The NPF Software Framework enables support for high availability by recognizing that the functions required to support HA functionality are provided by the SA Forum's Application Interface Specification (AIS) [9]. The NPF Software and High Availability (HA) section of this document illustrates how an NPF compliant network element can be extended with high availability functionality via the SA Forum AIS.

3.4.2 System Interconnects and Protocols

The NPF Software Framework has been generally designed to interoperate with interconnect protocols (i.e. chip-to-chip, FE-to-FE, or CE-to-FE), either proprietary or standardized. The NPF Software Framework assumes the presence of a protocol "engine" or "daemon", where required, to encode/decode protocol messages and transfer messages to the appropriate destination(s) within a network element. The one exception to this assumption is the NPF Messaging protocol, which is incorporated into the NPF Software Framework to enhance the Forwarding Plane usage model that the Framework defines.

3.4.2.1 IETF ForCES

The NPF recognizes the complementary nature of the NPF Software Framework and the efforts of the IETF ForCES working group [11]. The IETF Forwarding and Control Element Separation (ForCES) aims to define a framework and associated mechanisms for standardizing the exchange of information between logically separate functionality of the control plane. In the context of the NPF Framework, ForCES can be viewed as an interconnect protocol allowing the physical realization of a network element to be distributed amongst multiple cards/blades.

The NPF Functional API and LFB Model and the IETF ForCES FE Model share common semantics, and in most cases, operations with regards to LFB definitions. This commonality provides a basis for interoperability between NPF APIs and the ForCES protocol. In fact, the combination of NPF APIs and the IETF ForCES protocol provide the framework for composing network elements built from multi-vendor components (blades, devices, software, etc.) that interoperate along standard interfaces and protocols.

3.4.2.2 NPF Messaging

A NPF compliant network element may consist of NPEs from a variety of vendors. Ensuring proper communication, in both the vertical and horizontal, of NPEs is a primary objective of the NPF. The NPF has defined the NPF Messaging Implementation Agreement [7] for the exchange of datagrams between NPEs across a physical boundary. NPF Messaging can be used for both chip-to-chip communication as well as blade-to-blade communication. The NPF Software Framework further provides the means for managing the use of the Messaging protocol within a network element by defining a Messaging LFB in the forwarding plane and it's associated API [8].

There is a wide variety of system architectures in which there is a physical separation amongst system components. An example of this is where control plane and forwarding plane functionality are separated by a physical boundary (e.g. Ethernet, PCI Express or proprietary backplane), and the forwarding plane is distributed across multiple cards/blades. Figure 6 illustrates this architecture and highlights the interplay of NPF APIs (Service, Functional, and Interface Management), NPF Messaging and the IETF ForCES protocol. This figure is intended to provide clarity to the interconnect discussion and is not suggesting or recommending an implementation.

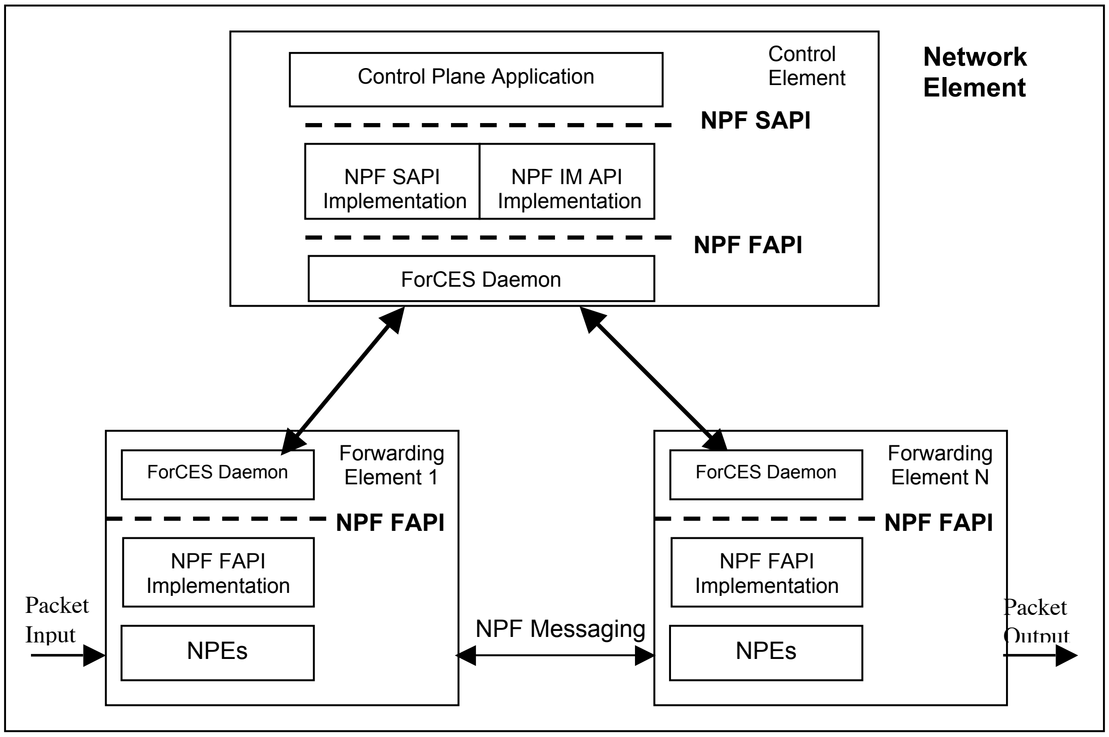

Figure 6 – Example NE with NPF APIs, NPF Messaging and IETF ForCES

4 Resource Management

NPF Software APIs use a callback based response model, i.e., most API requests return before the request is completed and the actual completion (or failure) of a request is communicated to the application via a separate callback function. This model is open to errors due to lost or duplicate callbacks. It is the application's responsibility to protect itself against duplicated or lost callbacks. The framework architecture assists an application in its recovery from lost callbacks and supports the resynchronization of state between the application and provider. The occurrence of lost or duplicate callbacks is considered a rare event and hence the performance of the recovery mechanism is not emphasized. The framework assumes that the application is able to detect both lost and duplicated callbacks.

The application should be able to detect duplicated callbacks and take the required steps to ignore the duplicates. The framework supports the use of application specified "correlator" on each API function call. The correlator can be used by the application to detect duplicated callbacks.

There are three categories of APIs that need to be considered in dealing with lost callbacks.

- **Resource Creation APIs**
  When a new resource is created, a handle for the resource is sent back to the application. If the callback is lost, the application cannot determine if the resource was actually created. Neither can the application delete this resource; since it does not have a handle for the resource; nor can the provider do garbage collection; since it does not know that the resource has been abandoned.

- **Resource Deletion APIs**
  A lost callback for a resource deletion API function does not pose a problem, since the function can be freely called again.

- **Resource Modification APIs**
  Loss of callbacks to API functions that make absolute changes does not pose a problem, since the function can be freely called again. However loss of callbacks to functions that make incremental changes do pose a problem to recovery.

The Framework supports the recovery of applications from lost callbacks by the following two methods

- Any API function that requests for the creation of new resource(s), supports a user-generated resource identifier. The provider retains the identifier for the existence of the resource. An application shall not be able to create two instances of the same resource type with the same id.
- Each API also provides query functions to list the resources created by the user and the contents and status of each resource.

5 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.

As previously mentioned, NPF Service APIs provide a "black box" view of the services provided, hiding both vendor and physical implementation details. NPF Service API implementations may interface with NPF Functional APIs to execute the service it provides. Additionally, Service API implementation may implement proprietary functions to carry out its service. If the use of proprietary functions necessitates a modification to the Service API, those modifications must appear in the form of proprietary extensions to the existing, NPF defined Service API.

NPF Service APIs cover a wide range of networking technologies such as IPv4 Forwarding, IPv6 Forwarding, DiffServ, MPLS, and IPSec. Details of all the technologies currently supported by NPF Service APIs can be found at [10]. The following sub-sections will outline some of the unique capabilities the NPF provides in support of the NPF Service APIs.

5.1 Services Discovery API (SDI)

The Services Discovery API (SDI) is used by the Control Plane to discover the services or capabilities being provided by a particular network element or by the Forwarding Plane on that network element. For example, on a Layer 7 switch, the SDI will be used to discover services such as TCP termination, SSL Acceleration, SLB (Server Load Balancing) being provided by the Forwarding Plane, depending on its functionality. Another example would be a VPN gateway which would expose services such as Stateful Firewall or SSL Acceleration. Once these services have been discovered, the Control Plane can use the corresponding Services API such as Secure Socket Layer (SSL) Service API or Server Load Balancer (SLB) Service API to configure and manage those services. The SDI component has been added to the NPF Services APIs in order to facilitate easy discovery and integration of advanced services defined by NPF.

6 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.

6.1 Interface Management API

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

The Interface management API provides a uniform interface for managing the attributes of interfaces (ports), either physical or logical, in the Forwarding Plane. The API allows for interface configuration, collection of interface statistics (e.g. number of transmission errors, number of dropped packets, etc.), and the establishment of relationships between interfaces (e.g. mapping of a Layer 2 interface to a Layer 3 interface).

The Interface Management API is actually a collection of API specifications consisting of a core set of API functions and a number of API extension specifications that address specific interface types (e.g. LAN, SONET/SDH, Packet over SONET, etc.). The family of Interface Management API specifications allows implementers to focus solely on management of those interfaces which are specific to their application.

The Interface Management API provides facilities for the collection of status and statistical data from interfaces. 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 appropriate Interface Management API specifications.

The Functional API (FAPI) defines software interfaces that are used below the Interface Management API for control of devices at a functional level. Use of FAPI/LFBs as part of an Interface Management API implementation helps to ensure multi-vendor compatibility and replaceability at the board or component level. As an example, optical framer vendors that supply an NPF defined Functional API will allow system integrators (including those that implement elements of the Interface Management API) the ability to select which device to use without having to factor in the software integration cost, as it is likely to be the same across all optical framer devices. Figure 7 provides an example of how NPF Functional APIs and the NPF Interface Management API may co-exist in a system.

Conversely, system integrators that do not require board or component replacement within a system may not require device vendors to supply NPF Functional APIs. These system integrators may still realize a benefit in requiring the NPF Interface Management API as the means for port management by control plane software.

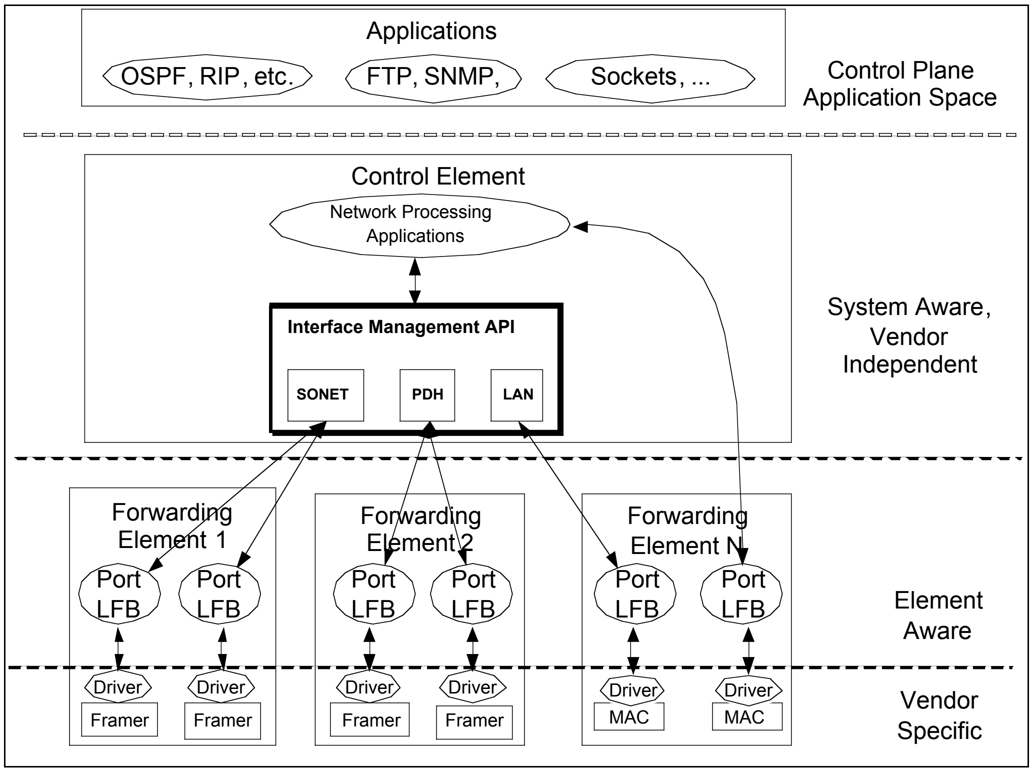

Figure 7 -- Example Interface Management API usage models

6.2 Packet Handler API

The NPF Packet Handler API provides a path for applications to send and receive packets through Forwarding Elements (FEs). 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 (see Figure 8).

The Packet Handler API can be used at multiple layers within a network element. At the system aware layer (Services layer) the Packet Handler API can support input and output sessions for individual applications, while at the element aware layer (Functional layer) it can be used to communicate directly with a Forwarding Element (FE). The use of the Packet Handler API at these two levels is conceptually analogous to the stated purposes of NPF Service APIs and NPF Functional APIs. The following diagram illustrates how the Packet Handler API can be applied at these different levels.

The implementations of the Packet Handler API may be layered, as illustrated in the figure below by “Forwarding Element N”. An example where such an implementation may be desired is when a Forwarding Element is comprised of multiple NPEs from different vendors, each of which supports its own version of the NPF Packet Handler API. From a system perspective, it may be advantageous to add another Packet Handler API on top of the vendor versions, thus providing a single entry/exit point for packets that traverse between the Control Element and the Forwarding Element.

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.

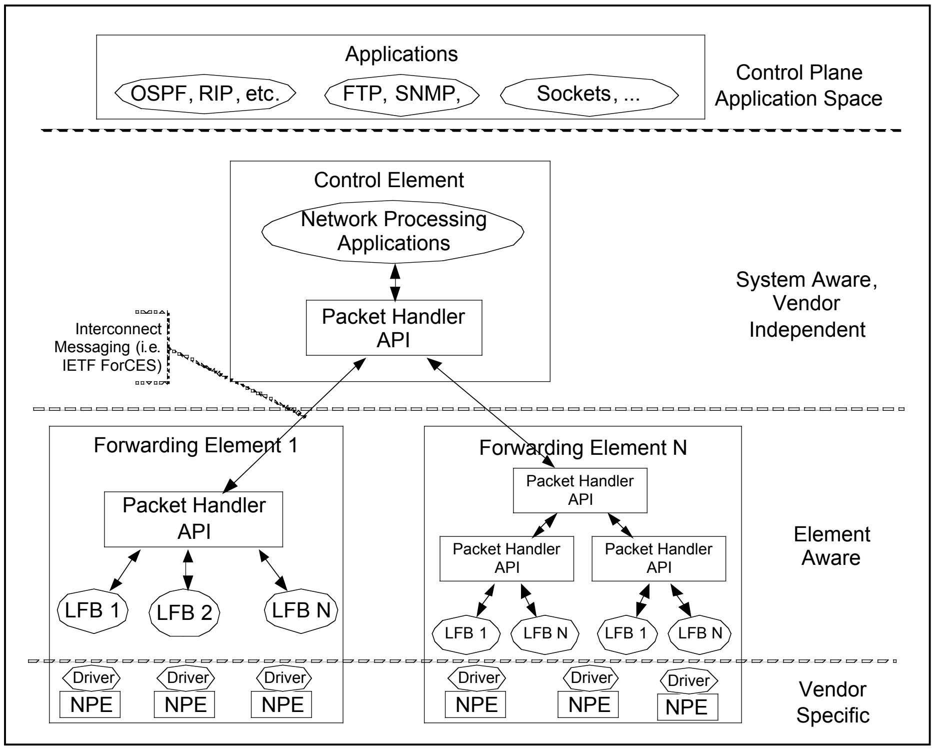

Figure 8 -- Packet Handler Architecture

System-aware code beneath the service level Packet Handler API de-multiplexes and dispatches packets between these clients and the functional Packet Handler APIs to individual NP devices.

7 NPF Functional APIs & LFB Model

The NPF defines a logical representation (a model) for networking processing functionality that may exist in the Forwarding Plane. Forwarding Plane functionality is quantified by the NPF into distinct articles of functionality (e.g. Forwarding, Metering, and Classification) and termed Logical Function Blocks (LFBs). Additionally, the NPF defines the APIs required to control/manage such functionality by elements in the Control and Management planes. These APIs are termed Functional APIs (FAPIs). NPF FAPIs are a set of industry defined APIs to be used to control/manage forwarding plane functionality, allowing the easy and rapid integration of network processing elements into any NPF-compliant system.

Potential beneficiaries of this concept are system and sub-system (i.e. blade) integrators. NPF FAPIs should ensure that the cost of integration is roughly independent of which device or device vendor is selected, thus the selection can be based on the packet processing features and performance of the candidate parts and not the cost of integration. Furthermore, by using industry defined Functional APIs, forwarding plane devices become fully interchangeable, eliminating the integrator's risk of being locked in to any given device or device vendor.

The key to the above promise is that when moving from one device to another, the rest of the system should require minimal changes, if any. There are two conditions which must be satisfied to achieve this. First, FAPIs must provide an unambiguous interface to control the forwarding plane. Vendor implementation differences that may affect the proper inter-operation between the forwarding and control planes must all be explicitly expressed via these APIs. The second condition is that only the FAPIs are used to control the forwarding plane. In other words, no vendor-specific APIs should be needed. FAPIs must present every relevant detail of the forwarding plane and provide all the tools required for their proper configuration.

The NPF Functional API model [4] provides a framework to permit interoperability of LFB implementations, which is achieved through specifying compatible syntax (when crossing NPE physical boundaries) and semantics of information exchanged between them (horizontally), configurable via the NPF Functional APIs (vertically). The NPF Functional API model is expected to be compatible with the ForCES Framework and Model that are being defined by the IETF ForCES working group. This model is used to describe the forwarding plane's functionality. The control plane may dynamically discover the functions (LFBs) 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 or not discovery occurs, the control plane uses the NPF Functional APIs to manipulate the functionality of the forwarding plane, represented by NPF defined LFBs.

7.1 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.

7.2 Functional Model

The NPF LFB model (or "model" for short) of the forwarding plane represents the functionality of the forwarding plane as a sequence of functional blocks termed Logical Function Blocks (LFBs). These LFBs represent the set of functions supported by a data plane, along with their ordering. The functional blocks may be arranged in a directed graph, with different LFBs applying to different subsets of packets. In such configurations an LFB may have multiple LFBs to which its outputs are "connected" delivering packets to a different downstream LFB based on its own internal classification of packets. These LFBs 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 LFB performs some well-specified logical function. Given that new types of network functionality evolve over time, the LFB model must reflect the existing state of the art and allow extensions to be specified to include new functions. LFBs 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 NPF LFB model does not prescribe the set of LFBs 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 LFB capabilities to be configured but, in the FE implementation itself, LFBs 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 LFB as eight individually named egress LFBs or as a single LFB with eight separate property capability sets.

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

7.2.1 The Configuration Manager Concept

Forwarding plane configuration may consist of managing the configuration of many LFBs. When there is a large number of LFBs to be individually managed for a FE, it may be expedient to provide a macro configuration mechanism for forwarding plane configuration. The NPF FAPI model allows for such a mechanism for FEs via the configuration manager LFB.

Using a connection oriented technology like ATM as an example [12], the process of setting up a single connection in the Forwarding Plane may require a number of API calls to various LFBs. If the LFBs required to setup a connection reside on the same Forwarding Element, however, it is possible to construct a "Configuration Manager" LFB that would serve as a proxy between the underlying LFBs and the Functional API client(s) that require access to the service provided by those LFBs. The "Configuration Manager" LFB exposes a Functional API that represents the aggregate functionality provided by the collection of LFBs, allowing FAPI clients to manage the functionality via a macro interface.

The implementation of the "Configuration Manager" LFB maintains the intelligence on how to provide the desired service (setting up an ATM connection in this example) working in concert with the underlying LFBs. How the "Configuration Manager" LFB implements the service is out of scope for the NPF Software Framework as the implementation may vary from vendor to vendor. Figure 9 depicts the concept of a "Configuration Manager" for ATM. The concept can be reused for any technology that requires a large number of LFBs, residing on the same FE, to manage a particular service/function.

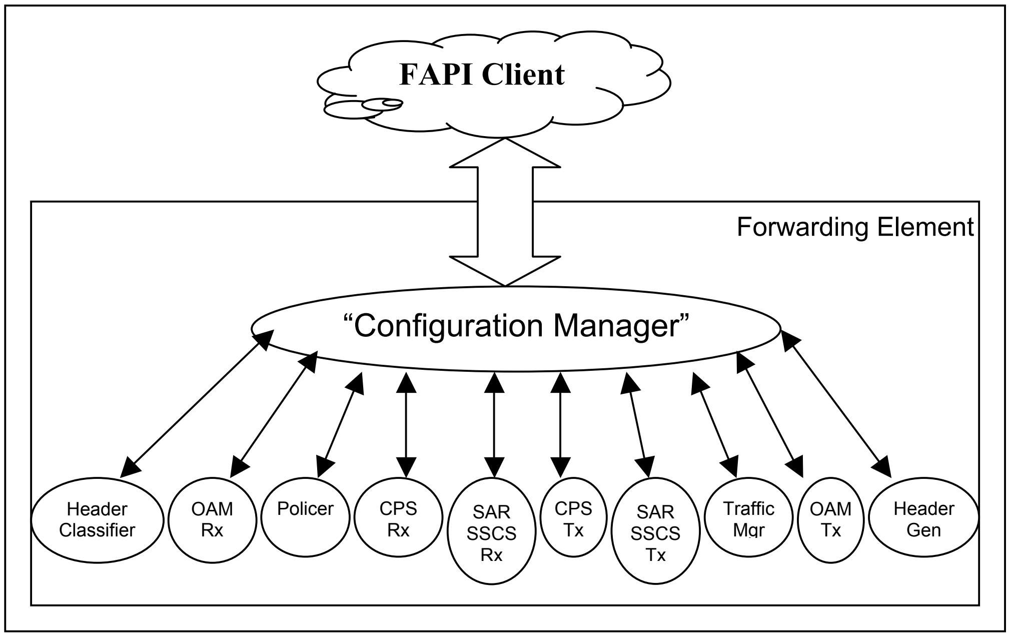

Figure 9 -- Example of an ATM "Configuration Manager"

An FE which is made up of a large number of LFBs may incorporate LFBs from multiple vendors. In such scenarios, it is possible to have "Configuration Manger" LFBs from each vendor offering. It is further possible to have the board integrator construct an additional "Configuration Manager" LFB so as to present a single interface for managing the FE. How these "Configuration Manager" LFBs are constructed, and how they share configuration information amongst LFBs is considered proprietary and out of scope for NPF.

7.3 Topology Discovery API

NPF Functional APIs are based on the model described in section 7.2 and consists of two parts. The first part provides methods for the manipulation of the capabilities of each LFB in the forwarding device. These are functional APIs that the NPF defines for a specific Forwarding Plane function (e.g. Classification, IPv4 Forwarding, etc.). The second part optionally provides a set of methods for the discovery of the LFBs, and their capabilities, supported by a forwarding element.

The method used for discovering LFBs and their capabilities is called topology discovery. The tool used to perform topology discovery is called the Topology Discovery API [6[gc1]]. The Topology Discovery API enables the functionality of a Forwarding Element to be comprehended by an application by discovering the LFBs that exist on the Forwarding Element. There are a number of scenarios in which implementation of the Topology Discovery API may be useful, one of them being when a system supports the insertion or removal of Forwarding Elements during system runtime (hot-plugging). In such a system, the insertion of a new Forwarding Element may cause the system to invoke the Topology Discovery API to find out what are the services provided by the new card and how the system can take advantage of those services. In fact, any system which supports aspects of dynamic configurability may require the Topology Discovery API.

7.4 Interconnecting FE LFB Topologies

The NPF LFB model defines a board (blade) level view of the Forwarding Plane, which may consist of several boards each with distinct functionality. It is expected that an entity outside the Forwarding Plane will be responsible for the coordination and management of the entire Forwarding Plane. The NPF has defined tools that facilitate interoperability of the various boards (in terms of the LFBs residing on a board) that may exist in an NPF compliant Forwarding Plane.

Two implementation agreements have been defined by the NPF to address NPE to NPE (or LFB Topology to LFB Topology) interoperability, namely the Messaging Layer IA [7] and the Messaging LFB IA [8]. The Messaging Layer IA specifies the format by which two NPF compliant HW components may exchange PDUs. The Messaging LFB IA specifies how one configures an NPE to transmit/receive specific PDU formats via an API. Implementation of these two specifications allows for "horizontal" interoperability between NPF compliant NPEs. Figure 12 illustrates the intended usage of these two specifications.

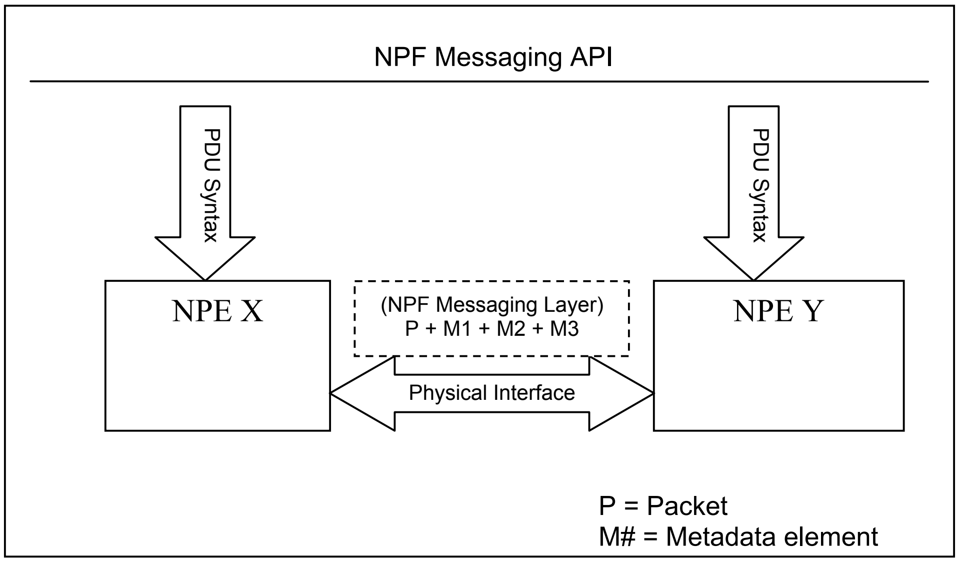

Figure 10~~12~~ -- NPF Messaging in the Forwarding Plane

8 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.

9 NPF Software and High Availability (HA)

The basis for specifying High Availability in the context of the NPF Software Framework is the Service Availability Forum's (SAF) Application Interface Specification (AIS) [9]. This section is a description of how AIS integrates into the NPF Software Framework. The NPF Software Framework does not attempt to define High Availability concepts, interfaces or mechanisms. Rather, the NPF Software Framework references the HA functionality defined by the SAF and illustrates here how the SAF AIS can be integrated into an NPF compliant system.

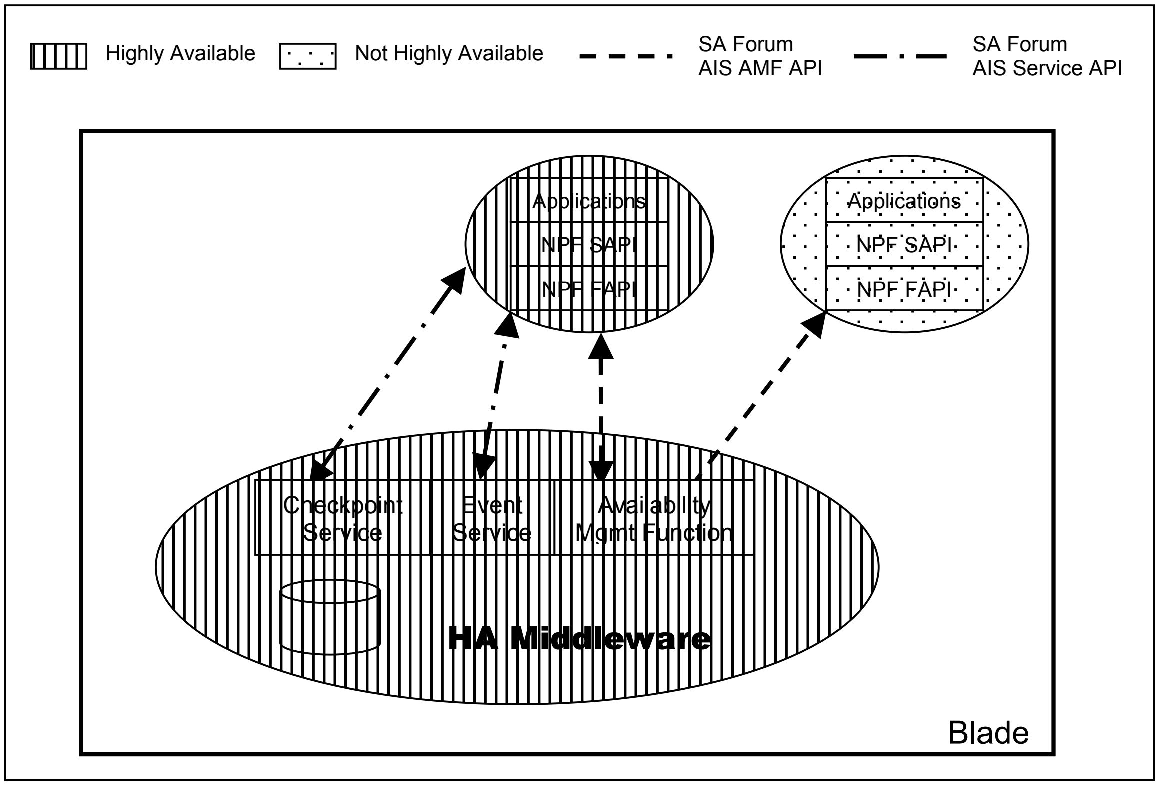

Figure 11~~13~~ -- HA conceptual model

Figure 13 provides a conceptual view of a highly available, NPF compliant blade within a network element, and it's HA relevant interfaces. The figure depicts the major components that may materialize in a line card or control card and will likely be replicated amongst multiple cards in a highly available network element. The HA Middleware is a distributed cluster framework for communicating state and availability information between HA-aware applications and resources (including NPF Service API and FAPI implementations) in a collection of nodes. Additionally, the HA Middleware on a node will periodically perform health checks on the HA-aware applications and resources to ensure that it has the most current state information.

Depending on the implementation, the HA Middleware may only be able to perform basic operations on non-HA applications, such as Start and Stop operations.

The HA Middleware is responsible for managing all HA related functions for a single card/blade. There will be an HA Middleware process for each card/blade in a network element. Each card containing a HA Middleware process will discover the presence of other cards containing HA Middleware and synchronize the states of resources across cards. The HA Middleware is responsible for coordinating switch over activities if a failure should occur.

An HA Middleware exposes interfaces that are to be used for HA management, as depicted in Figure 13. Two of these interfaces have relevance in the NPF Software Framework and are defined as part of the SAF AIS API [9].

1. **SA Forum AIS Availability Management Framework (AMF) API**: The AMF API provides functions for registration/de-registration, health monitoring, state maintenance, resource management and event/error reporting.
2. **SA Forum AIS Service API:** The AIS Service API provides HA related services to HA-aware applications such as checkpointing (for replication management) and event services (for notification of status change).

9.1 SA Forum AIS

In order to provide continuous service to applications, NPEs may provide a set of High Availability services and functions. The NPF recognizes the SA Forum AIS as an appropriate method for providing HA functionality. Figure 14 provides a pictorial representation of the high availability interface reference points defined by the SA Forum, namely the Application Management Function API (Hm), the HA Service API (Hs), and the HA Protocol Interface (Hp). These reference points are the interfaces that applications/services will use to maintain high availability. Figure 14 also depicts, for completeness, the NPF APIs (SAPI, FAPI, Interface Management, Packet Handler) as a library that applications/services use. The intent of the figure is to show the relationship of all these components in a system and is not intended to suggest an implementation.

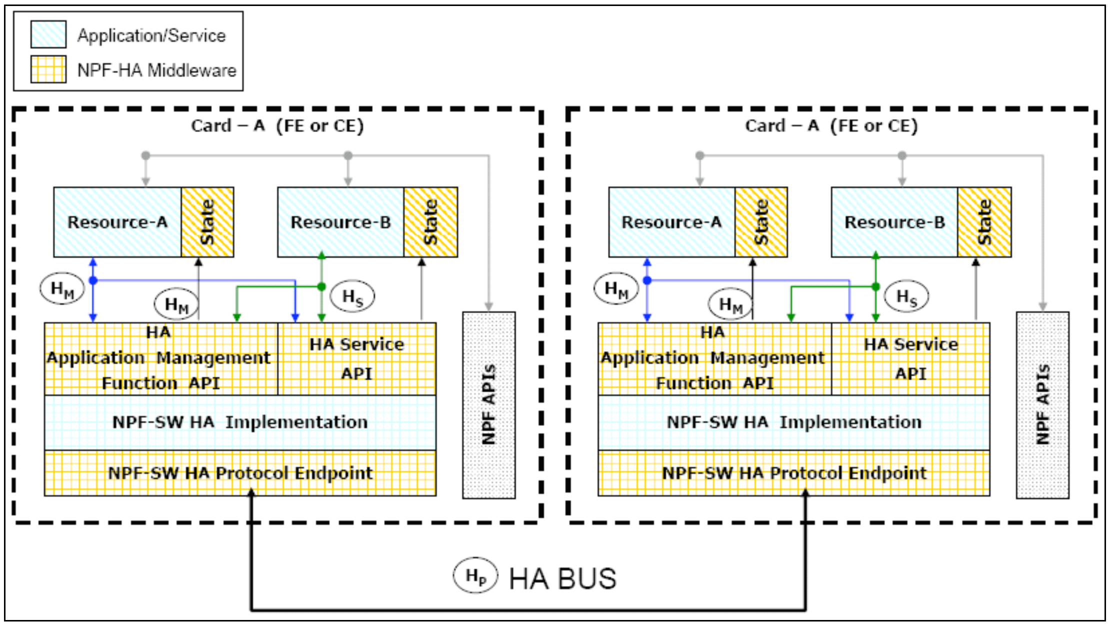

Figure 12~~14~~ HA reference points

9.1.1 AIS Application Management Framework API (Hm):

The AIS Availability Management Function (AMF) API, or Hm, is an interface to the HA-aware middleware, and is part of the SAF AIS. The functions contained in the AMF API support the following functions for each HA-aware component.

- Registration and De-registration with the HA-aware middleware
- Monitoring health by invoking callbacks and maintaining HA state
- Reporting events and errors

The Hm interface provides both synchronous as well as asynchronous functions using callbacks.

9.1.2 AIS Service API (Hs):

The AIS Service API, or Hs, is a collection of APIs that provide High Availability services to its clients such as checkpointing, event management, and cluster membership. Currently only event and checkpoint services are identified as relevant HA services for the NPF Software Framework. The NPF Software Framework allows for extensions for future HA services.

9.1.3 HA Protocol Interface (Hp):

The HA Protocol Interface, or Hp, is used by HA implementations to provide continuous service operation and replication across cards/blades. The Hp interface is an open interface to exchange messages across Cards within a network element, and could use messaging protocols such as NPF Messaging or SA Forum's messaging API. Currently, the Hp interface specification is outside the scope of NPF.

9.2 Adding High Availability to NPF APIs

In order to make NPF Functional and Service API implementations HA-aware, the implementations will need to be extended to make calls to the HA Middleware, using the SAF AIS. The SAF AIS will be used by NPF API implementations to register HA resources, transfer state information and pass event information to the HA Middleware. These additions to NPF API implementations must in no way restrict existing functionality of the NPF API. Additionally, the NPF API exposed to NPF API clients must not change in order to support high availability. However, the clients may have to be modified to interact with HA middleware for initialization and event polling. The NPF FAPI and Service API implementations that implement these additional API calls are called HA-aware FAPIs (FAPI-H) and HA-aware Service APIs (SAPI-H) respectively or generally termed HA-aware NPF APIs (NPF API-H).

9.2.1 HA interaction with NPF APIs

Figure 15 describes the interactions between a HA-aware application, HA Middleware and HA-aware NPF APIs (NPF API-H) and is used to illustrate how NPF APIs and the SAF AIS can co-exist in a highly available system.

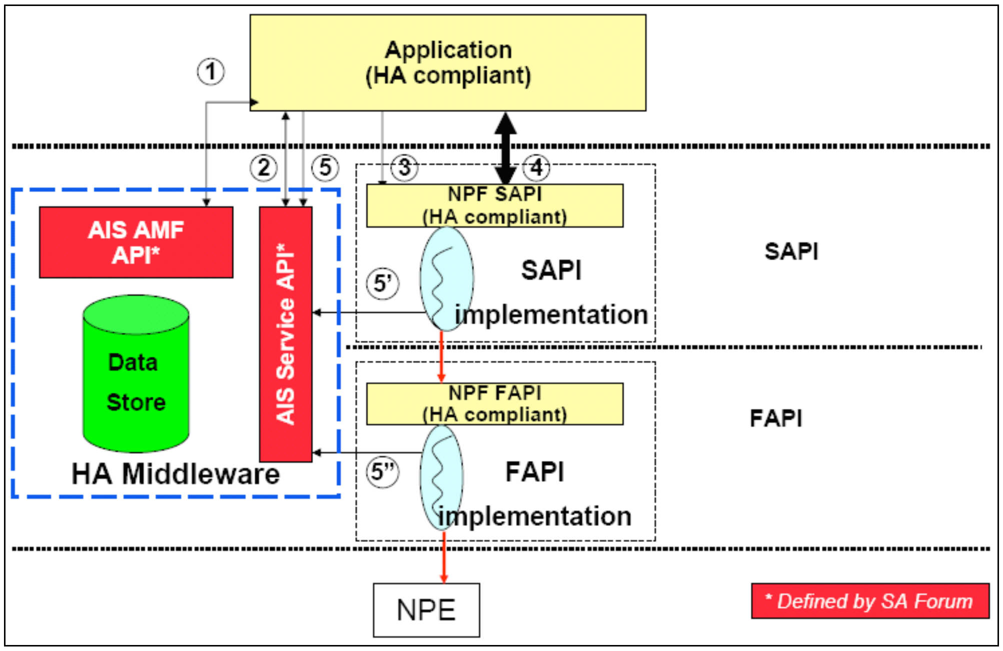

Figure 13~~15~~ -- NPF and HA components

Figure 15 also describes the sequence of interactions between an HA-aware application that uses NPF API-H and the HA Middleware.

1. The HA-aware application registers with the HA Middleware under a unique component name. It also passes the instance identifier (e.g., process identifier) during the registration process.
2. If the HA-aware application maintains states and wishes to use the HA Service API, it can register with one or more HA services. It does so by performing registration for each HA service (i.e. checkpoint, event logging).
3. After successful registration and initialization, the HA-aware application calls the NPF API-H initialization routine(s) and passes HA context information to the NPF API-H. This information will be later used by the NPF API-H implementation to invoke HA services on behalf of the application.
4. The HA-aware application periodically listens for messages from HA middleware using mechanisms like socket select (a preferred mechanism for SAF programming model).
5. The HA-aware application invokes NPF API-H function calls to perform its function/service.
6. The NPF API-H, after completing the normal processing, invokes HA Service APIs as needed to perform tasks such as update state information or log events with the HA-aware middleware.
7. The HA-aware application processes any HA message received from the peer like state update or transaction update.
8. The HA-aware application listens to switchover message and initiates switchover action upon receiving the message. Steps 4 through 8 are repeated.

10 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.

11 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.

4 Munro, Alistair (ed.), NPF Functional API Model and Usage Guidelines Implementation Agreement, http://www.npforum.org/techinfo/fapi_npf2002.340.38.pdf

5 Khosravi, Hormuzd (ed.), NPF Topology Discovery Functional API Implementation Agreement, http://www.npforum.org/techinfo/topology_fapi_npf2002%20438%2023.pdf

6 Johnson, Erik (ed.), NPF Messaging Layer Implementation Agreement, October 2003, http://www.npforum.org/techinfo/Messaging_IA.pdf

7 Cain, Gamil (ed.), NPF Messaging LFB Implementation Agreement, September 2004, http://www.npforum.org/techinfo/messaging_lfb_npf2003.504.13.pdf

8 Service Availability Forum, Application Interface Specification (AIS), http://www.saforum.org/specification/AIS%20Information

9 NPF Implementation Agreements, http://www.npforum.org/techinfo/approved-chronological.shtml

10 IETF ForCES WG Supplemental Home Page, http://www.sstanamera.com/~forces/index.html

11 Herneld, Patrik (ed.), NPF ATM Software Architecture Framework Implementation Agreement, April 2005, http://www.npforum.org/techinfo/npf2004.088.12.pdf

12 Kumar, Vinoj N. (ed.), NPF Software API Framework Lexicon, http://www.npforum.org/techinfo/LexiconIAv1.pdf

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

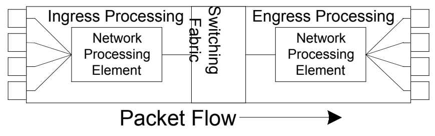

Figure 14~~16~~. Network processing element hardware configurations

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 17 shows an example of this, where a NPE is dedicated to each ingress and each egress framer.

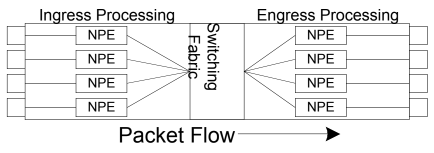

Figure 15~~17~~. Network processing element per port

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 18 is an example of such a configuration where there might be multiple network processing elements in use in the several of the chassis.

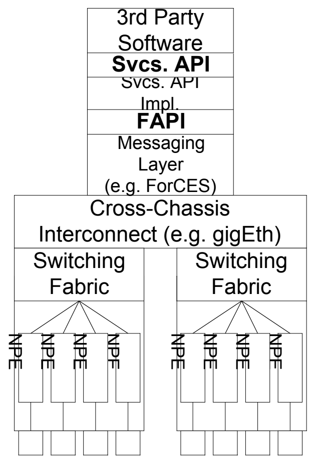

Figure 16~~18~~. Example multi-chassis NPF architecture

Appendix B. Functional API Examples

B.1 IPv4 Forwarding Blade Functional API Examples

This section describes how the NPF 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.

B.1.1 Exact-Match IPv4 Forwarding with Static Configuration

In this example, a blade is being integrated into a system using NPF Functional APIs. 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 LFBs – ingress, IPv4 forwarding, and egress. The ingress LFB represents eight fast Ethernet ports, the IPv4 forwarder LFB performs exact match forwarding and requires a next hop MAC address, and that the egress LFB represents eight fast Ethernet ports. Furthermore, the IPv4 forwarder LFB exposes events that represent IPv4 forwarding table misses.

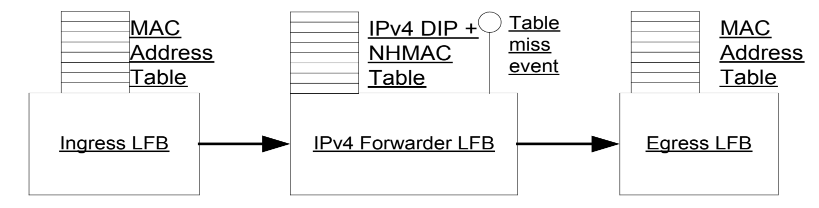

Figure 17~~19~~. IPv4 Forwarding Functional Model Example #1

Using the LFB information and APIs provided in vendor data sheets, the control plane configures the three LFBs 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. The 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 LFB that classifies packets and creates metadata that informs the egress LFB 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 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.

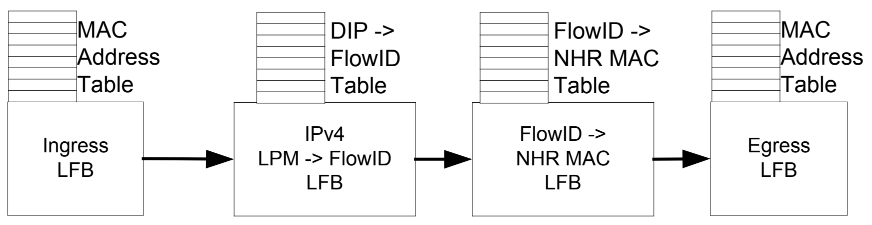

Figure 18~~20~~. IPv4 Forwarding Blade Functional Model Example #2

B.1.2 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 LFBs 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 NPF topology discovery API [6] provided, the control plane discovers that the forwarding blade is represented as four LFBs. The ingress LFB represents eight fast Ethernet ports, the IPv4 forwarder LFB performs longest prefix match forwarding and associates a flowid with each packet, a next hop resolution LFB indexes the flowID to a next hop router and its MAC address, and an egress LFB represents eight fast Ethernet ports.

Appendix C. Protocol Data Unit Processing

The following sub-sections provide examples of the kinds of PDU processing that may occur in the Forwarding Plane of a network element.

C.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.

C.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.

C.3 LAN Bridging (Ethernet Switching) Processing

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)

C.4 ATM Switching

ATM switching information includes.

- Per port VPI/VCI forwarding databases.

Appendix D. Acknowledgements

Working Group Chair: Alex Conta

Task Group Chair: Gamil Cain

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

The list is in alphabetical order of last names:

Gamil Cain, Intel

Ajay Kamalvanshi, Nokia

Jayaram Kutty, IDT

Alistair Munro, U4EA

Michael Persson, Ericsson

John Renwick, Agere Systems

Per Wollbrand, Ericsson

Appendix E. List of companies belonging to NPF during approval process

Agere Systems

AMCC

Analog Devices

Cypress Semiconductor

Enigma Semiconductor

Ericsson

Flextronics

Freescale Semiconductor

HCL Technologies

Hifn

IDT

Infineon Technologies AG

Intel

IP Fabrics

IP Infusion

Motorola

Mercury Computer Systems

Nokia

NTT Electronics

PMC-Sierra

Sensory Networks

Sun Microsystems

Teja Technologies

TranSwitch

U4EA Group

Wintegra

Xelerated

Xilinx