Binding and Discovery

Design Specification

Control Plane-Platform Development Kit 2.11

March 2004

Contents

Figures

Tables

Revision History

Revision	Description	Date	Author
2.11	Updated for Release 2.11	March 2004	Udaya Shankar
2.1	Updated for Release 2.1	December 2003	Udaya Shankar
2.0	Updated for Release 2.0	August 2003	Udaya Shankar

Part 1: Introduction

1 Introduction

Network elements such as switches and routers can be classified into three logical operational components:

- Control plane
- Forwarding plane
- Management plane

The control plane controls and configures the forwarding plane and the forwarding plane manipulates the network traffic. The control plane executes different signaling or routing protocols and provides all the routing information to the forwarding plane.

The forwarding plane makes decisions based on this information and performs operations on packets such as forwarding, classification, filtering, and so on.

An orthogonal management plane manages the control and forwarding planes. For example, the control plane in a router executes routing protocols, the forwarding plane performs hardware-based switching, and the management plane starts or stops routing process, and performs logging.

The introduction of standardized Application Program Interface (API) within the above-mentioned planes can help system vendors, Original Equipment Manufacturer (OEM), and end-users of these network elements to mix and match components available from different vendors to achieve a device of their choice. The Network Processing Forum (NPF) API is designed for this purpose, as it presents a flexible and well-known programming interface to the control plane applications. It makes the existence of multiple forwarding planes, as well as vendor-specific details, transparent to control plane applications.

The hardware properties and nature of interconnect used between the control and the forwarding planes are isolated. Thus, the protocol stacks and network processors available from different vendors can be easily integrated with the NPF APIs. The APIs included in the Control Plane Platform Development Kit (CP-PDK) are based on the NPF APIs. For more information about NPF, refer to http://www.npforum.org/.

This document specifies the design of the binding and discovery module implemented for the CP-PDK. The binding and discovery module is responsible for discovering the capabilities of the underlying forwarding planes when they bind to the control plane. This module exposes uniform semantics for the discovery of the forwarding planes, their resources, and capabilities to the other CP-PDK modules.

1.1 Purpose/Scope

This document provides information needed to implement and/or maintain the binding and discovery module of the CP-PDK.

1.2 Terminology

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

Table 1. Terminology table

Term	Description
B&D	Binding and Discovery
C&M	Configuration and Management
CE	Control Element
CP-PDK	Control Plane Platform Development Kit
FE	Forwarding Element
ForCES	Forwarding and Control Element Separation protocol
IXA	Internet Exchange Architecture
NPF	Network Processing Forum
PDK	Platform Development Kit

1.3 References

Table 2 lists documents referenced in, or related to, this document. All of the documents listed in this table are included with the CP-PDK.

Table 2. Reference table

Reference	Document Name
[1]	NPF Namespace Specification, NP Forum
[2]	CP-PDK API Framework Reference
[3]	CP-PDK Software Architecture Overview
[4]	CP-PDK Forwarding Plane Plugin API Reference
[5]	CP-PDK Transport Plugin Design Reference
[6]	CP-PDK Configuration and Management API Reference

Part 2: Overview

2 Overview

The binding and discovery module is responsible for determining the capabilities of newly bound FEs and adding them to the namespace.

2.1 High-Level Functionality Overview

FE establishes connections to the control plane after performing any FE-specific initialization, such as, initializing the core components in the Intel® Internet Exchange Architecture (IXA) platform. The manner in which the connections are established and maintained is beyond the scope of this document. Connection establishment and maintenance is transport dependent and must be handled accordingly. These details are addressed by the transport plug-in component of the PDK.

Figure 1 shows how the binding and discovery module fits into the CP-PDK. For more details on the CP-PDK, see CP-PDK Software Architecture Overview [3].

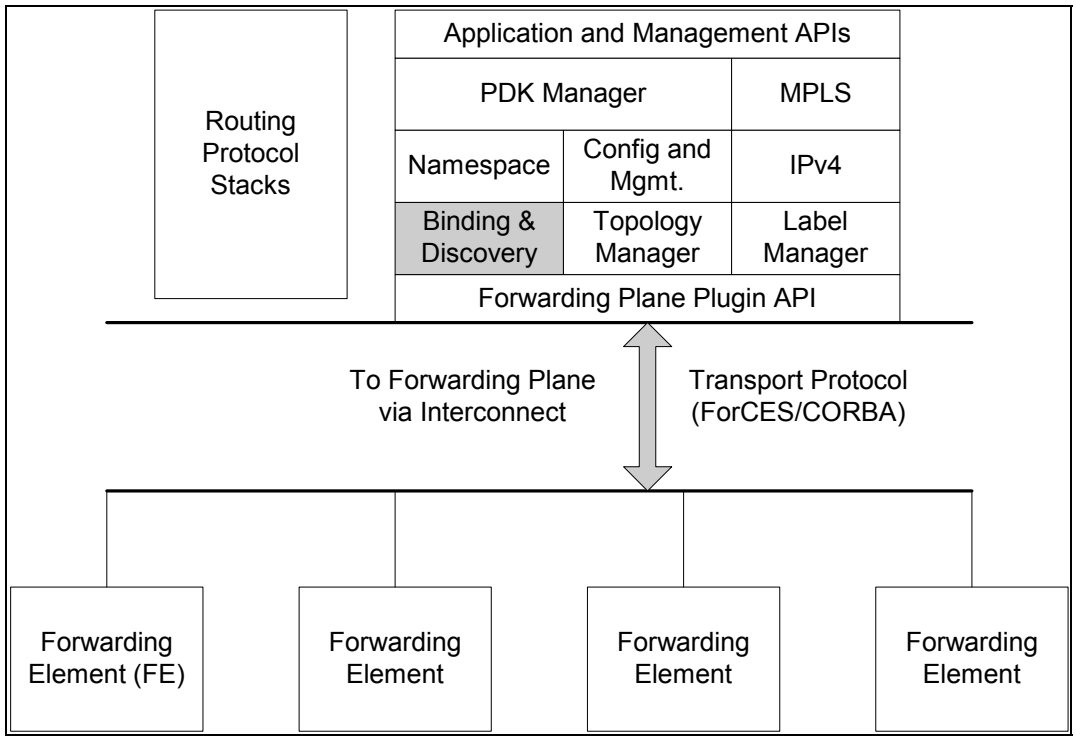

Figure 1: Binding and discovery module in CP-PDK

In the PDK architecture, the control and forwarding planes are physically separated by an interconnect, such as, Ethernet and PCI. The FE binding process involves establishing connections between the transport plug-in of the control plane part of the PDK and the individual FEs. After FE identification, the FE reports its capabilities to the control plane. This includes forwarding capabilities, number of available ports, individual port capabilities, etc. The binding and discovery module obtains these FE properties and capabilities and informs relevant control plane components about them.

This module interacts with and populates the namespace module with appropriate objects, such as, FE and ports, based on the learned capabilities of the forwarding planes.

Note The process of binding and capability discovery can be dynamic. The process supports hot-swap ability, known as dynamic binding, of the FEs can dynamically bind and unbind to the system. B&D keeps record of FEs from earlier binds so that namespace, C&M nodes, and resources created during a previous bind can be reused.

The process of binding and capability discovery of the forwarding plane consists of three distinct events:

- FE binding to the CE through the transport plug-in: Individual forwarding elements might be connected to the control plane across different transport mechanisms and protocols. For example, the CE and FE might be separated by shared memory or a physical interconnect such as Ethernet. The protocol used for CE-FE communication could be a standard protocol such as Transport Communication Protocol/Internet Protocol (TCP/IP) or ForCES, or a proprietary protocol that determines exactly how the FE binds to the CE. For example, in the case of a simple TCP/IP-based protocol, the FE might bind to a server socket created on the CE that waits for connection requests from FEs. If the communication protocol being used was ForCES or a proprietary protocol, connection establishment might occur after a sequence of FE identification and authentication.
- FE capability discovery: Once the CE-FE connection is established, the FE reports its capabilities to the CE. This also depends on the protocol used to exchange information between the CE and FE. For example, the FE can report all its capabilities, such as, FE ID, number of ports, and individual port capabilities, through one method invocation. The other possibility is that the CE and FE engage in information exchange that would lead to the CE obtaining FE capability information over a sequence of information exchanges. The exact manner in which this information is exchanged and obtained depends on the protocol used.
- Reporting bind event and FE capability to the PDK: Once the transport plug-in has registered a newly bound FE and discovered its capabilities, this must be reported to the other PDK modules. This can be done in a manner independent of the underlying transport plug-in and communication protocol used. This functionality is termed as binding and discovery. This document details the design of this component of the PDK and its interactions with other PDK modules and is termed B&D. The location of this module within the PDK is illustrated in Figure 1.

The FE binding to the CE through the transport plug-in and FE capability discovery events depend on the transport plug-in. FE and CE use a communication protocol to interact with each other. To maintain portability of the PDK, the above implementation-specific details must be hidden within the transport plug-in. This part of the B&D process is termed vendor-specific B&D and is a part of the transport plug-in implementation. Further details on this are specified as part of the transport plug-in module.

2.2 Design Considerations, Assumptions, and Dependencies

- This module depends on the FP plug-in API implementation to deliver FE bind events.
- Binding is a 2-step process:
    - FP plug-in reports that an FE is trying to bind. This module replies with a Bind response.

- FP plug-in passes in all the information required – FE and its capabilities, all in one message.

The bind process may be different, such as, an exchange of multiple messages between this module and the FE before all the capability information is exchanged. The 2-step process is a choice made in the CP-PDK. It is the responsibility of the FP plug-in API to adhere to this requirement. If the underlying capability reporting protocol changes, the B&D module should still be aware of only these two steps.

Part 3: Component Design

3 Component Design

3.1 High-Level Overview

The binding and discovery module interacts with the PDK in the following ways:

- **Interaction with the FP plug-in API:** The binding and discovery module registers callbacks with the FP plug-in API for receiving notifications:
  - When a new FE binds and reports its capabilities
  - When the FE reports individual port capabilities
- **Interaction with the namespace:** When the binding and discovery module discovers that a new FE has bound to the system, it must populate the namespace module. The exact mechanism and protocol used to bind an FE and discover its capabilities is handled by the vendor-specific binding and discovery module within the transport plug-in. It must:
  - Create a namespace node corresponding to the new FE and obtain an NPF handle to the new FE node.
  - Check if FE port nodes are also created in the namespace and basic properties learned from the bind event are configured.
- **Interaction with C&M:** After creating the required FE and port nodes in the namespace, use the NPF handle to populate the FE's properties and capabilities learned from the bind event.

3.2 Dependencies

The B&D module currently requires a few features from other components of the PDK:

- The transport plug-in must send binding and capability discovery information to this module.
- The namespace and C&M components must allow this module to add placeholders for the binding of forwarding planes and their corresponding capabilities.

3.3 External API

External APIs are called PDK manager and are described in 3.3.1 and 3.3.2 sections.

3.3.1 Initialization

The PDK manager initializes this module during PDK startup. Since this module registers callbacks with the FP plug-in module and allows C&M to register for callbacks, it must be initialized after the FP plug-in and before C&M has been initialized.

```
NPF_RET bd_init();
```

3.3.2 Shutdown

The PDK manager shuts down this module during PDK shutdown. It must be shut down before the FP plug-in module.

```
NPF_RET bd_shutdown();
```

3.4 Implementation Details

This section describes the design of the B&D module.

3.4.1 Execution Context

The interaction between B&D and other modules of the PDK is explained in the following contexts:

- During initialization, the PDK manager initializes B&D. B&D executes and registers callbacks with the FP plug-in module to receive notifications of FE bind events and FE/Port capability report events.
- The FP plug-in invokes callbacks registered by B&D when an FE binds and reports its capabilities/attributes. B&D creates new namespace nodes for the new FE and its ports and calls back C&M to report an FE bind.

3.4.2 Initialization

The PDK manager initializes the B&D module during PDK startup. Since this module registers callbacks with the FP plug-in module and allows C&M to register for callbacks, it must be initialized after FP plug-in initialization and before C&M has been initialization.

bd_init()

```
{
```

Initialize context for registering callback - `bd_context`

register `NPF_FE_BIND` event callback with `FPAPI`

```
fpapi_event_register_callback(bd_context, NPF_FE_BIND);
```

register `PORT_CAPABILITY` event callback with `FPAPI`

```
fpapi_event_register_callback(bd_context, NPF_PORT_CAP);
}
```

3.4.3 Shutdown

The PDK manager shuts down the B&D module during PDK shutdown. This module must be shut down before the FP plug-in module.

bd_shutdown

```
{
        deregister callback for FPPI_FEBind_Event

        deregister callback for FPPI_FEPortCap_Event
}
```

3.4.4 Module Use Scenarios

This module is functional only during the FE bind process and is responsible for initializing all FE-related data in the PDK modules, such as, namespace and C&M. This includes all capabilities of the FE and its individual ports.

Note: This data does not change at runtime, but can only be configured. Once a port reports its FE and port capabilities, such as congestion control capabilities, queuing, etc., it is then configured.

Line speed, Maximum Transmission Unit (MTU), and IP address are set by an administrative application. Administrative applications can possibly alter the properties of a port by changing its line speed, MTU, or other parameters. These, however, do not mean a change in port capability. Capabilities like Virtual Local Area Network (VLAN) and link aggregation are configurable properties and do not mean a change in capabilities.

3.4.4.1 FE Bind Event

On receiving an FE Bind event from the FP plug-in, B&D replies with a successful bind response. This indicates to the FE that the bind was successful and it can go ahead and report its capabilities.

NoteThis is generic as this model can support any kind of negotiation/information exchange that might be done by the transport plug-in before an FE successfully binds to the CE.

OnFEBind(FPPI_FEID fe_id)

```
{
      report SUCCESS to FE.
      FPPAPI_feBindReply (fe_id, FPPI_SUCCESSFUL);
}
```

3.4.4.2 FE/Port Capability Event

The sequence of events that occur during an FE/Port Capability event are shown in Figure and explained following the figure. This use case is the first interaction that occurs between PDK

components after an FE bind event and it results in creation of new placeholders for the new FE and its ports in the namespace and C&M.

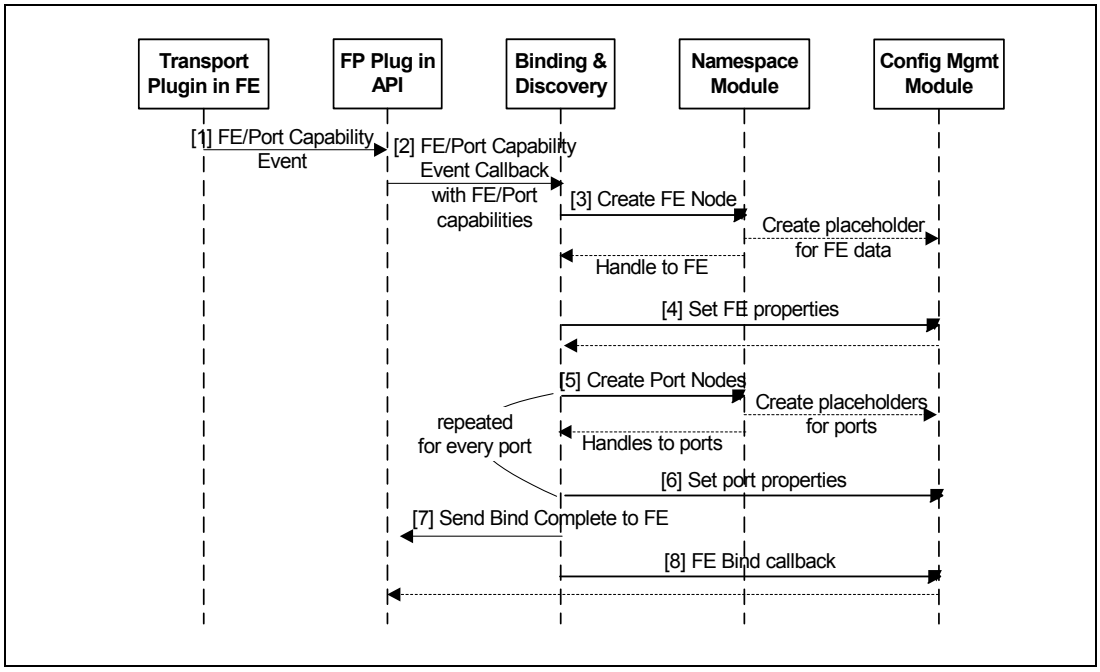

Figure 2: Sequence of events that occur during an FE/Port capability event

1. An FE binds to the CE and reports its capabilities, including individual port capabilities. The exact details of the mechanism of the protocol used and the manner in which capabilities are reported, are addressed by the transport plug-in.
2. The FP plug-in API now invokes the FE bind callback registered by B&D. The callback additionally contains parameters, such as, the FE ID, FE-wide capabilities, and other FE-wide attributes, like model number.
3. B&D now invokes the namespace API to create a new FE node in the namespace hierarchy. The FE bind is not reported to external applications or other internal modules till B&D has completed initializing FE-related data in the namespace and C&M modules. This call internally ensures that a placeholder is created for storing and maintaining the FE data. A namespace handle is returned to B&D corresponding to the newly created FE node in the namespace.

The name supplied by the FE identifying itself may be truncated to remove the:

- Leading non-numeric characters, and
- All characters following the end of the first contiguous set of numeric characters.

For Example, FE1 becomes 1, FE23 becomes 23, FE2.3 becomes 2 and FE3-2 becomes 3. The truncated name should be unique and is used to populate the namespace.

4. Using the namespace handle, B&D acquires an NPF handle to the new FE object and initializes any FE data received during the bind event. This includes data such as FE ID, model number, etc.
5. B&D invokes the namespace API to create corresponding port nodes in the namespace hierarchy under the FE node to which the port belongs. This internally creates a placeholder for the port data in the C&M module. The FP plug-in API receives

information about the port capabilities of the FE. In the current implementation, each FE reports its individual port capabilities in one shot. Since this is dependent on the protocol used, the FP plug-in API always exposes the same behavior to the B&D module. B&D always expects port capabilities of an individual FE to arrive through a single callback. The FP plug-in API takes care of aggregating this information if the underlying protocol changes.

6. B&D now acquires an NPF Handle to the port object and initializes any port-specific information in the placeholder for the port data just created.

Note: Steps 5 and 6 are repeated for every port on the FE.

7. Initialization is now complete. B&D sends a bind complete message to the FE through the FP plug-in API. This results in sending a message to the FE indicating that the bind process was completed successfully on the CE. The bind complete message must be sent to the FE before invoking the C&M module callback. The reason for this is that, if the C&M callback is invoked first, it will result in C&M invoking an application callback. If the application begins making C&M API calls to configure the FE properties, these API calls might all fail, since the FE has not yet completed the binding process. If the FE binding process is completed before invoking any callback, subsequent calls to configure and manage the FEs will probably succeed.
8. B&D invokes the callback registered by the C&M module indicating that a new FE has bound to the CE and necessary initialization has been performed.

This sequence of events completes the binding and discovery process of a new FE.

```
On_FEPortCap_Event (FPPI_FEID fe_id, void* response)
{
```

In params: FE ID,(pointer to struct of) FE capabilities

create node in Namespace - instance node, `TYPE = FE`, `parent = SYSTEM`

open the created node

now obtain the `NPF HANDLE` corresponding to the node created

```
npf_ns_getDataHandle(fe_ns_handle);
```

set FE properties using internal API

```
cm_fe_set_attribs (fe_data_handle, FE_attribs);
```

for (all ports) {

create port node in Namespace (`TYPE = PORT, parent = fe_handle`)

```
port_ns_handle = npf_ns_create(fe_handle, NPF_PORT);
```

open the created node

```
npf_ns_open(port_ns_handle);
```

now obtain the NPF HANDLE corresponding to the node created

```
NPF_HANDLE port_data_handle =
npf_ns_getDataHandle(port_ns_handle);
```

set port properties using internal API

```
cm_port_set_attribs (port_handle, port_cap_list[index])
}
```

Now that all NS nodes have been initialized, report bind event to CM

```
}
```

3.4.5 Internal API

The FP plug-in API defines the following functions:

- Callback function for FE bind event
- Callback function for port capability event

The only other API function exposed by binding and discovery is the registration and deregistration functions used by the C&M module for registering/de-registering for the FE bind event. Since the C&M module is the only consumer for B&D, the registration and de-registration functions are simple, they do not require the contexts and callback handles associated with the normal API callbacks.

3.4.5.1 Registering Callback for FE Bind Event

Syntax

```
NPF_RET bd_bind_register_callback (IN: BD_BIND_CALLBACK
fe_bind_callback );
```

Description

This function is used by the C&M module to register a callback function for receiving notification when a new FE binds.

Input Parameters

`BD_BIND_CALLBACK` Callback function invoked when the event occurs.

3.4.5.2 Deregistering Callback for FE Bind Event

Syntax

```
NPF_RET bd_bind_deregister_callback ();
```

Description

The C&M module uses this function to de-register a callback function.

3.4.5.3 Callback for FE Bind Event

Syntax

```
typedef void (*BD_BIND_CALLBACK) ( IN: NPF_HANDLE fe_handle );
```

Description

This is the callback function registered by C&M module.

Input Parameters

NPF_HANDLE The NPF handle to the newly created FE object (object handle)

3.4.6 Data Structures

All data structures used by B&D are defined by the FP plug-in API [4] and C&M API [6].