Internet Draft
Expiration: April 2004
File: draft-ietf-forces-model-01.txt
Working Group: ForCES

L. Yang
Intel Labs
J. Halpern
Megisto Systems
R. Gopal
Nokia
A. DeKok
IDT Inc.
Z. Haraszti
S. Blake
Ericsson

October 2003

ForCES Forwarding Element Model

draft-ietf-forces-model-01.txt

Status of this Memo

Abstract

This document defines the forwarding element (FE) model used in the Forwarding and Control Plane Separation (ForCES) protocol. The model represents the capabilities, state and configuration of forwarding elements within the context of the ForCES protocol, so that control elements (CEs) can control the FEs accordingly. More specifically, the model describes the logical functions that are

present in an FE, what capabilities these functions support, and how these functions are or can be interconnected. This FE model is intended to satisfy the model requirements specified in the ForCES requirements draft [1]. A list of the basic logical functional blocks (LFBs) is also defined in the LFB class library to aid the effort in defining individual LFBs.

Table of Contents

Conventions used in this document

The key words "MUST", "MUST NOT", "REQUIRED", "SHALL", "SHALL NOT", "SHOULD", "SHOULD NOT", "RECOMMENDED", "MAY", and "OPTIONAL" in this document are to be interpreted as described in [RFC-2119].

1. Definitions

A set of terminology associated with the ForCES requirements is defined in [1] and is not copied here. The following list of terminology is relevant to the FE model defined in this document.

FE Model -- The FE model is designed to model the logical processing functions of an FE. The FE model proposed in this document includes three components: the modeling of individual logical functional blocks (LFB model), the logical interconnection between LFBs (LFB topology) and the FE level attributes including FE capabilities. The FE model provides the basis to define the information elements exchanged between the CE and the FE in the ForCES protocol.

Datapath -- A conceptual path taken by packets within the forwarding plane, inside an FE. There might exist more than one datapath within an FE.

LFB (Logical Function Block) class (or type) -- A template representing a fine-grained, logically separable and well-defined packet processing operation in the datapath. LFB classes are the basic building blocks of the FE model.

LFB (Logical Function Block) Instance -- As a packet flows through an FE along a datapath, it flows through one or multiple LFB instances, with each implementing an instance of a certain LFB class. There may be multiple instances of the same LFB in an FE's datapath. Note that we often refer to LFBs without distinguishing between LFB class and LFB instance when we believe the implied reference is obvious for the given context.

LFB Model -- The LFB model describes the content and structures in LFB and associated data definition. There are four types of information defined in the LFB model. The core part of the LFB model is LFB class definitions while the other three are to define the associated data including common data types, supported frame formats and metadata.

LFB Metadata -- Metadata is used to communicate per-packet state from one LFB to another, but is not sent across the network. The FE model defines how such metadata is identified, produced and consumed by the LFBs, but not how metadata is encoded within an implementation.

LFB Attribute -- Operational parameters of the LFBs that must be visible to the CEs are conceptualized in the FE model as the LFB attributes. The LFB attributes include, for example, flags, single parameter arguments, complex arguments, and tables that the CE can read or/and write via the ForCES protocol.

LFB Topology -- Representation of how the LFB instances are logically interconnected and placed along the datapath within one

FE. Sometimes it is also called intra-FE topology, to be distinguished from inter-FE topology. LFB topology is outside of the LFB model, but part of the FE model.

FE Topology -- Representation of how the multiple FEs in a single NE are interconnected. Sometimes it is called inter-FE topology, to be distinguished from intra-FE topology (i.e., LFB topology). Individual FE may not have the global knowledge of full FE topology, but the local view of its connectivity with other FEs are considered part of the FE model. FE topology is discovered by the ForCES base protocol or some other means.

Inter-FE Topology -- See FE Topology.

Intra-FE Topology -- See LFB Topology.

LFB class library -- A set of LFB classes that are identified as the most common functions found in most FEs and hence should be defined first by the ForCES Working Group.

2. Introduction

[2] specifies a framework by which control elements (CEs) can configure and manage one or more separate forwarding elements (FEs) within a networking element (NE) using the ForCES protocol. The ForCES architecture allows Forwarding Elements of varying functionality to participate in a ForCES network element. The implication of this varying functionality is that CEs can make only minimal assumptions about the functionality provided by FEs in a NE. Before CEs can configure and control the forwarding behavior of FEs, CEs need to query and discover the capabilities and states of their FEs. [1] mandates that the capabilities, states and configuration information be expressed in the form of an FE model.

RFC 3444 [11] made the observation that information models (IMs) and data models (DMs) are different because they serve different purposes. "The main purpose of an IM is to model managed objects at a conceptual level, independent of any specific implementations or protocols used". "DMs, conversely, are defined at a lower level of abstraction and include many details. They are intended for implementors and include protocol-specific constructs." Sometimes it is difficult to draw a clear line between the two. The FE model described in this document is first and foremost an information model, but it also has a flavor of a data model as it contains explicit definition of the LFB class schema and other data structures. It is expected that this FE model will be used as the basis to define the payload for information exchange between the CE and FE in the ForCES protocol.

2.1. Requirements on the FE model

[1] defines requirements which must be satisfied by a ForCES FE model. To summarize, an FE model must define:

- Logically separable and distinct packet forwarding operations in an FE datapath (logical functional blocks or LFBs);
- The possible topological relationships (and hence the sequence of packet forwarding operations) between the various LFBs;
- The possible operational capabilities (e.g., capacity limits, constraints, optional features, granularity of configuration) of each type of LFB;
- The possible configurable parameters (i.e., attributes) of each type of LFB;
- Metadata that may be exchanged between LFBs.

2.2. The FE Model in Relation to FE Implementations

The FE model proposed here is based on an abstraction of distinct logical functional blocks (LFBs), interconnected in a directed graph, and receiving, processing, modifying, and transmitting packets along with metadata. Note that a real forwarding datapath implementation should not be constrained by the model. On the contrary, the FE model should be designed such that different implementations of the forwarding datapath can all be logically mapped onto the model with the functionality and sequence of operations correctly captured. However, the model itself does not directly address the issue of how a particular implementation maps to an LFB topology. This is left to the forwarding plane vendors as to how the FE functionality is represented using the FE model. Nevertheless, we do strive to design the FE model such that it is flexible enough to accommodate most common implementations.

The LFB topology model for a particular datapath implementation MUST correctly capture the sequence of operations on the packet. Metadata generation (by certain LFBs) must always precede any use of that metadata (by subsequent LFBs in the topology graph); this is required for logically consistent operation. Further, modifications of packet fields that are subsequently used as inputs for further processing must occur in the order specified in the model for that particular implementation to ensure correctness.

2.3. The FE Model in Relation to the ForCES Protocol

The ForCES base protocol is used by the CEs and FEs to maintain the communication channel between the CEs and FEs. The ForCES protocol may be used to query and discover the inter-FE topology. The

details of a particular datapath implementation inside an FE including the LFB topology, along with the operational capabilities and attributes of each individual LFB, are conveyed to the CE within information elements in the ForCES protocol. The model of an LFB class should define all of the information that would need to be exchanged between an FE and a CE for the proper configuration and management of that LFB.

Definition of the various payloads of ForCES messages (irrespective of the transport protocol ultimately selected) cannot proceed in a systematic fashion until a formal definition of the objects being configured and managed (the FE and the LFBs within) is undertaken. The FE Model document defines a set of classes and attributes for describing and manipulating the state of the LFBs of an FE. These class definitions themselves will generally not appear in the Forces protocol. Rather, Forces protocol operations will references classes defined in this model, including relevant attributes (and operations if such are defined).

Section 9 provides more detailed discussion on how the FE model should be used by the ForCES protocol.

2.4. Modeling Language for FE Model

Even though not absolutely required, it is beneficial to use a formal data modeling language to represent the conceptual FE model described in this document and a full specification will be written using such a data modeling language. Using a formal language can help in enforcing consistency and logical compatibility among LFBs. In addition, formal definition of the LFB classes has the potential to facilitate the eventual automation of some part of the code generation process and the functional validation of arbitrary LFB topologies.

The modeling language is used for writing the specification but not necessarily for encoding the data over-the-wire between FEs and CEs. When selecting the specification language, human readability is very important, while there are no performance requirements on the language for encoding, decoding, and transmission on the language. XML is used as the specification language in this document, because XML has the advantage of being human and machine readable with widely available tools support.

The encoding method for over the wire transport is an issue independent of the specification language chosen here. It is outside the scope of this document and up to the ForCES protocol to define.

2.5. Document Structure

Section 3 provides conceptual overview of the FE model, laying the foundation for the more detailed discussion and specifications in the sections that follow. Section 4, 5, and 6 together constitute the core of the FE model, detailing the three major components in the FE model: LFB model, LFB topology, and FE level attributes including capability. Section 7 presents a list of LFB classes in the LFB class library that will be further specified according to the FE model presented in earlier Sections (4, 5 and 6). Section 8 directly addresses the model requirements imposed by the ForCES requirement draft [1] while Section 9 explains how the FE model should be used in the ForCES protocol.

3. FE Model Concepts

Some of the most important concepts used throughout this document are introduced in this section. Section 3.1 explains the difference between a state model and a capability model, and how the two can be combined in the FE model. Section 3.2 introduces the concept of LFBs (Logical Functional Blocks) as the basic functional building blocks in the FE model. Section 3.3 discusses the logical inter-connection and ordering between LFB instances within an FE, that is, the LFB topology.

The FE model proposed in this document is comprised of these three components: LFB model, LFB topology and FE attributes including FE capabilities. The LFB model provides the content and data structures to define each individual LFB class; LFB topology provides a mean to express the logical inter-connection between the LFB instances along the datapath(s) within the FE; and FE attributes provide information at the FE level and the capabilities about what the FE can or cannot do at a coarse level. Details on each of the three components are described in Section 4, 5 and 6, respectively. The intention of this section is to discuss these concepts at the high level and lay the foundation for the detailed description in the following sections.

3.1. State Model and Capability Model

The FE capability model describes the capabilities and capacities of an FE in terms of variations of functions supported or limitations contained. Conceptually, the FE capability model presents the many possible states allowed on an FE with capacity information indicating certain quantitative limits or constraints.

For example, an FE capability model may describe the FE at a coarse level such as:

- this FE can handle IPv4 and IPv6 forwarding;
- this FE can perform classification on the following fields: source IP address, destination IP address, source port number, destination port number, etc;
- this FE can perform metering;
- this FE can handle up to N queues (capacity);
- this FE can add and remove encapsulating headers of types including IPSec, GRE, L2TP.

On the other hand, an FE state model describes the current state of the FE, that is, the instantaneous values or operational behavior of the FE. The FE state model presents the snapshot view of the FE to the CE. For example, using an FE state model, an FE may be described to its CE as the following:

- on a given port the packets are classified using a given classification filter;
- the given classifier results in packets being metered in a certain way, and then marked in a certain way;
- the packets coming from specific markers are delivered into a shared queue for handling, while other packets are delivered to a different queue;
- a specific scheduler with specific behavior and parameters will service these collected queues.

The information on the capabilities and capacities of the FE helps the CE understand the flexibility and limitations of the FE functions, so that the CE knows at a coarse level what configurations are applicable to the FEs and what are not. Where it gets more complicated is for the capability model to cope with the detailed limits, issues such as how many classifiers the FE can handle, how many queues, and how many buffer pools the FE can support, how many meters the FE can provide.

While one could try to build an object model for representing capabilities in full, other efforts have found this to be a significant undertaking. A middle of the road approach is to define coarse-grained capabilities and simple capacity measures. Then, if the CE attempts to instruct the FE to set up some specific behavior it is not capable of, the FE will return an error indicating the problem. Examples of such approach include Framework Policy Information Base (PIB) [RFC3318) and Differentiated Services QoS Policy Information Base [4]. The capability reporting classes in the DiffServ and Framework PIBs are all meant to allow the device to indicate some general guidelines about what it can or cannot do, but do not necessarily allow it to indicate every possible

configuration that it can or cannot support. If a device receives a configuration that it cannot implement, it can reject such configuration by replying with a failure report.

Figure 1 shows the concepts of FE state, capabilities and configuration in the context of CE-FE communication via ForCES protocol.

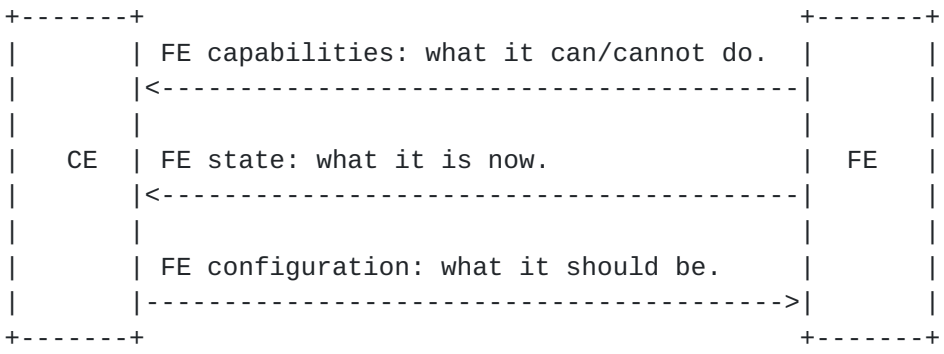


Figure 1. Illustration of FE state, capabilities and configuration exchange in the context of CE-FE communication via ForCES.

The ForCES FE model must include both a state model and some flavor of a capability model. We believe that a good balance between simplicity and flexibility can be achieved for the FE model by combining the coarse level capability reporting with the error reporting mechanism. Examples of similar approach include DiffServ PIB [4] and Framework PIB [5].

The concepts of LFB and LFB topology will be discussed in the rest of this section. It will become clear that some flavor of capability model is needed at both the FE level and LFB level.

Capability information at the LFB level is an integral part of the LFB model, and is modeled the same way as the other operational parameters inside an LFB. For example, certain features of an LFB class may be optional, in which case it must be possible for the CE to determine if an optional feature is supported by a given LFB instance or not. Such capability information can be modeled as a read-only attribute in the LFB instance. See Section 4.4.4 for more details on LFB attributes.

Capability information at the FE level may describe what LFB classes the FE can instantiate; how many instances of each can be created; the topological (i.e., linkage) limitations between these LFB instances, etc. Section 6 defines the FE level attributes including capability information.

Once the FE capability is described to the CE, the FE state information can be represented by two levels. The first level is the logically separable and distinctive packet processing functions, and we call these individual functions Logical Functional Blocks (LFBs). The second level of information is about how these individual LFBs are ordered and placed along the datapath to deliver a complete forwarding plane service. The interconnection and ordering of the LFBs is called LFB Topology. Section 3.2 discuss high level concepts around LFBs while Section 3.3 discuss issues around LFB topology.

3.2. LFB Modeling

Each LFB (Logical Functional Block) performs a well-defined action or computation on the packets passing through it. Upon completion of such function, either the packets are modified in certain ways (like decapsulator, marker), or some results are generated and stored, probably in the form of metadata (like classifier). Each LFB typically does one thing and one thing only. Classifiers, shapers, meters are all examples of LFB. Modeling LFB at such fine granularity allows us to use a small number of LFBs to create the higher-order FE functions (like IPv4 forwarder) precisely, which in turn can describe more complex networking functions and vendor implementations of software and hardware.

(Editor's note: We need to revisit the granularity issue around LFB later and provide a practical design guideline as how to partition the FE functions into LFB classes. We will gain more insight on the subject once we debate and settle on the LFB list in the LFB class library, described in Section 7. So the text around granularity here might be revised to reflect the lessons we learn.)

An LFB has one or more inputs, each of which takes a packet P, and optionally metadata M; and produces one or more outputs, each of which carries a packet P', and optionally metadata M'. Metadata is data associated with the packet in the network processing device (router, switch, etc.) and passed between one LFB to the next, but not sent across the network. It is most likely that there are multiple LFBs within one FE, as shown in Figure 2, and all the LFBs share the same ForCES protocol termination point that implements the ForCES protocol logic and maintains the communication channel to and from the CE.

An LFB, as shown in Figure 2, has inputs, outputs and attributes that can be queried and manipulated by the CE indirectly via Fp reference point (defined in [2]) and the ForCES protocol

termination point. The horizontal axis is in the forwarding plane for connecting the inputs and outputs of LFBs within the same FE. The vertical axis between the CE and the FE denotes the Fp reference point where bidirectional communication between the CE and FE happens: the CE to FE communication is for configuration, control and packet injection while the FE to CE is for packet re-direction to the control plane, monitoring and accounting information, errors, etc. Note that the interaction between the CE and the LFB is only abstract and indirect. The result of such interaction is for the CE to indirectly manipulate the attributes of the LFB instances.

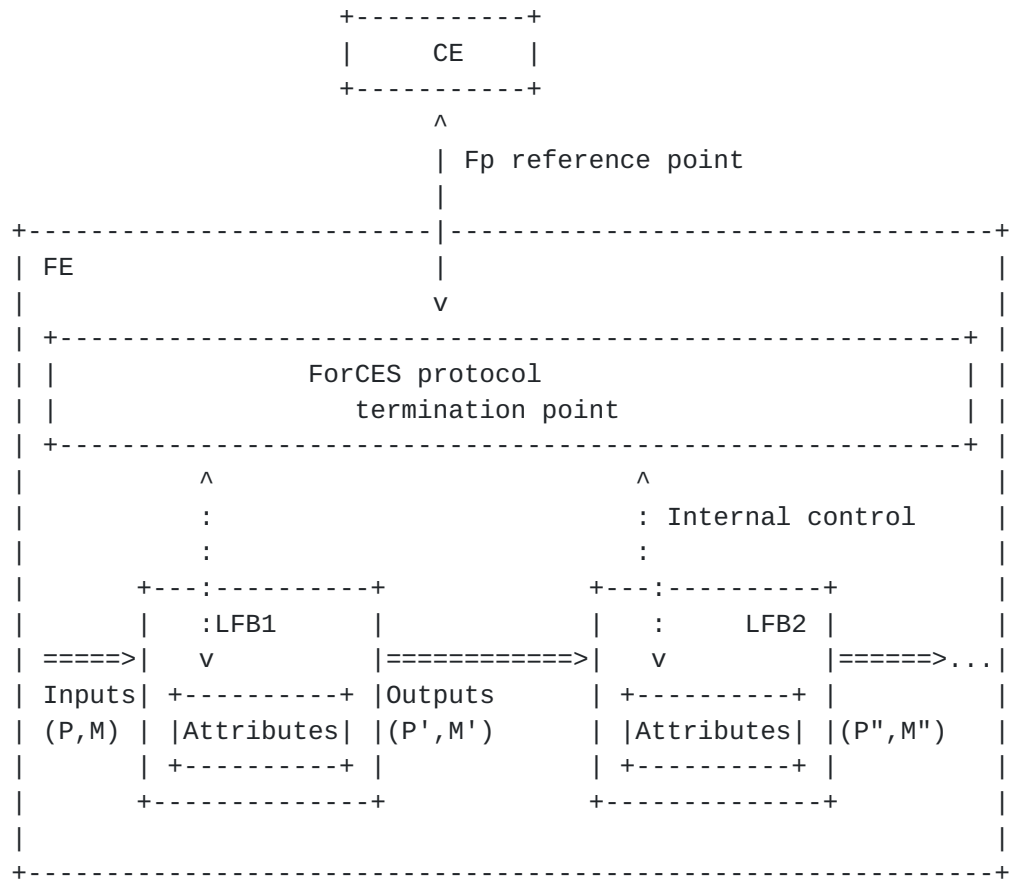


Figure 2. Generic LFB Diagram

A namespace is used to associate a unique name or ID with each LFB class. The namespace must be extensible so that new LFB class can also be added later to accommodate future innovation in the forwarding plane.

LFB operation must be specified in the model to allow the CE to understand the behavior of the forwarding datapath. For instance, the CE must understand at what point in the datapath the IPv4 header TTL is decremented (i.e., it needs to know if a control packet could be delivered to the CE either before or after this point in the datapath). In addition, the CE must understand where and what type of header modifications (e.g., tunnel header append or strip) are performed by the FEs. Further, the CE must verify that various LFB along a datapath within an FE are compatible to link together.

There is value to vendors if the operation of LFB classes can be expressed in sufficient detail so that physical devices implementing different LFB functions can be integrated easily into a FE design. Therefore, semi-formal specification is needed; that is, a text description of the LFB operation (human readable), but sufficiently specific and unambiguous to allow conformance testing and efficient design (i.e., eliminate guess-work), so that interoperability between different CEs and FEs can be achieved.

The LFB class model specifies information like:

- number of inputs and outputs (and whether they are configurable)
- metadata read/consumed from inputs;
- metadata produced at the outputs;
- packet type(s) accepted at the inputs and emitted at the outputs;
- packet content modifications (including encapsulation or decapsulation);
- packet routing criteria (when multiple outputs on an LFB are present);
- packet timing modifications;
- packet flow ordering modifications;
- LFB capability information;
- LFB operational attributes, etc.

Section 5 of this document provides detailed discussion on the LFB model with a formal specification of LFB class schema. The rest of Section 3.2 here only intends to provide conceptual overview of some important issues in LFB modeling, without covering all the specific details.

3.2.1. LFB Input and Input Group

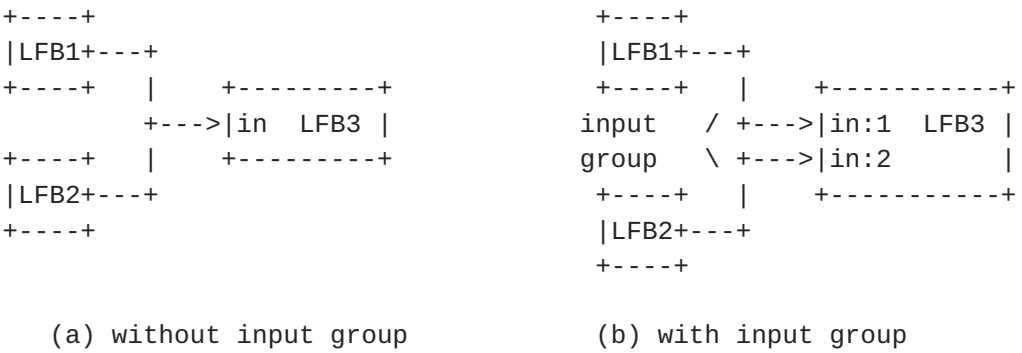
An LFB input is a conceptual port of the LFB where the LFB can receive information from other LFBs. The information is typically a packet (or frame in general) and associated metadata, although in

some cases it might consist of only metadata, i.e., with a Null-packet.

It is inevitable that there will be LFB instances that will receive packets from more than one other LFB instances (fan-in). If these fan-in links all carry the same type of information (packet type and set of metadata) and require the same processing within the LFB, then one input should be sufficient. If, however, the LFB class can receive two or more very different types of input, and the processing of these inputs are also very distinct, then that may justify the definition of multiple inputs. But in these cases splitting the LFB class into two LFB classes should always be considered as an alternative. In intermediate cases, e.g., where the inputs are somewhat different but they require very similar processing, the shared input solution should be preferred. For example, if an Ethernet framer LFB is capable of receiving IPv4 and IPv6 packets, these can be served by the same LFB input.

Note that we assume the model allows for connecting more than one LFB output to a single LFB input directly. There is no restriction on the number of up-stream LFBs connecting their outputs to the same input of a single LFB instance. Note that the behavior of the system when multiple packets arrive at such an input simultaneously is not defined by the model. If such behavior needs to be described, it can be done either by separating the single input to become multiple inputs (one per output), or by inserting other appropriate LFBs (such as Queues and possibly Schedulers) between the multiple outputs and the single input.

If there are multiple inputs with the same input type, we model them as an input group, that is, multiple instances of the same input type. In general, an input group is useful to allow an LFB to differentiate packet treatment based on where the packet came from.



(a) without input group (b) with input group

Figure 3. An example of using input group.

Consider the following two cases in Figure 3(a) and (b). In Figure 3(a), the output from two LFBs are directly connected into one input of LFB3, assuming that it can be guaranteed no two packets arrive at the same time instance. If LFB3 must do something different based on the source of the packet (LFB1 or LFB2), the only way to model that is to make LFB1 and LFB2 to pass some metadata with different values so that LFB3 can make the differentiation based on the metadata. In Figure 3(b), that differentiation can be elegantly expressed within LFB3 using the input group concept where the instance id can server as the differentiating key. For example, a scheduler LFB can potentially use an input group consisting of a variable number of inputs to differentiate the queues from which the packets are coming.

3.2.2. LFB Output and Output Group

An LFB output is a conceptual port of the LFB where it can send information to some other LFBs. The information is typically a packet (or frame in general) and associated metadata, although in some cases it might emit only metadata,, i.e., with a Null-packet.

We assume that a single LFB output can be connected to only one LFB input (this is required to make the packet flow through the LFB topology unambiguous). Therefore, to allow any non-trivial topology, multiple outputs must be allowed for an LFB class. If there are multiple outputs with the same output type, we model them as output group, that is, multiple instances of the same output type. For illustration of output group, consider the hypothetical LFB in Figure 4. The LFB has two types of outputs, one of which can be instantiated to form an output group.

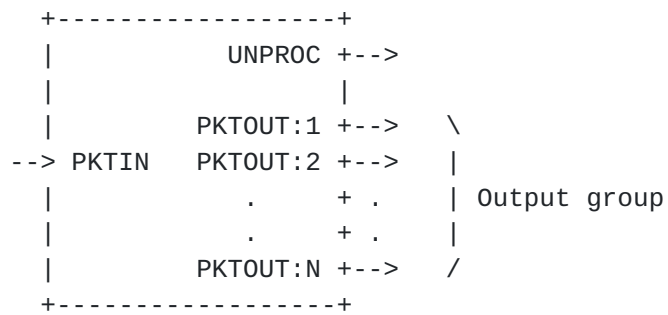


Figure 4. An example of an LFB with output group.

Multiple outputs should mainly be used for functional separation where the outputs are connected to very different types of LFBs. For example, an IPv4 LPM (Longest-Prefix-Matching) LFB may have one default output to send those packets for which look-up was

successful (passing a META_ROUTEID as metadata); and have another output for sending packets for which the look-up failed. The former output may be connected to a route handler LFB, while the latter can be connected to an ICMP response generator LFB or to a packet handler LFB that passes the packet up to the CE.

3.2.3. Packet Type

When LFB classes are defined, the input and output packet formats (e.g., IPv4, IPv6, Ethernet, etc.) must be specified: these are the types of packets a given LFB input is capable of receiving and processing, or a given LFB output is capable of producing. This requires that distinct frame types be uniquely labeled with a symbolic name and/or ID.

Note that each LFB has a set of packet types that it operates on, but it does not care about whether the underlying implementation is passing a greater portion of the packets. For example, an IPv4 LFB might only operate on IPv4 packets, but the underlying implementation may or may not be stripping the L2 header before handing it over -- whether that is happening or not is opaque to the CE.

3.2.4. Metadata

Metadata is used to communicate per-packet state from one LFB to another. To ensure inter-operability among LFBs, the LFB class specification must define what metadata the LFB class "reads" or "consumes" on its input(s) and what metadata it "produces" on its output(s). For that purpose, metadata types must be identified. For example, an META_IFID, passed from a port LFB to an IPv4 processing LFB (with the IP packet) can be one of the defined metadata types.

Symbolic names can be assigned for common metadata types. In addition, additional information such as numeric data type, maximum and minimum accepted values, and special values should be defined for each metadata value. Some of these constraints will be defined in the LFB class model, and some of them may be specific capabilities of a particular LFB instance.

While it is important to define the metadata passing between LFB in terms of its name, value and interpretation, it is not necessary to define the exact encoding mechanism used by LFBs for metadata. Different implementations are allowed to use different encoding mechanisms for metadata. For example, one implementation may store

metadata in registers or shared memory, while another implementation may encode metadata in-band as preamble in the packets.

A given LFB may require a certain metadata at its inputs for its internal processing. What should happen with the metadata after it is read by the LFB? In particular, should the metadata be propagated along with the packet when the packet is forwarded from the LFB to the next LFB, or should it be removed (consumed) by the LFB?

In certain cases, passing the metadata along is desirable. For example, a META_CLASSID metadata may denote the result of a classification LFB and used in more than one downstream LFBs to trigger the proper operation on the packet. In this case the first LFB that uses the META_CLASSID should also allow the META_CLASSID to be passed with the packet to the next LFB, and so on. On the other hand, it is easy to see that if metadata is never consumed by LFBs, then as the packet trickles through the datapath, a large number of metadata will potentially be accumulated by the packet.

We believe that one way to accommodate both scenarios is to specify the propagation mode for each element of metadata utilized by an LFB class. Metadata elements which are not propagated are specified with the CONSUME mode, while elements which are propagated are specified with the PROPAGATE mode.

However, whether a metadata is useful beyond an LFB may depend on the actual LFB topology, i.e., what other LFBs are placed downstream. So the propagation mode of metadata should be configurable.

A packet may arrive to an LFB with metadata that is not meaningful to that LFB, but may be important to some other downstream LFBs. To cater to such cases it should be the assumed (default) behavior of all LFB classes that they transparently propagate any metadata elements that they do not utilize internally.

Actual implementations of LFBs in hardware may have limitations on how much metadata they can pass through. The limitation may be expressed in terms of total framesize (packet + metadata), metadata total size, number of metadata elements, or a combination of these. The limitation may be on the FE level or may be specific to LFBs within an FE. The pass-through capabilities of LFB instances and FEs can be queried as part of the capability discovery process.

(Editor's note: The definition of metadata here is only preliminary and the authors intend to work on the subject in more detail. Input is most welcome.)

3.2.5. LFB Versioning

LFB class versioning is a method to enable incremental evolution of LFB classes. Unlike inheritance (discussed next in Section 3.2.6), where it assumed that an FE datapath model containing an LFB instance of a particular class C could also simultaneously contain an LFB instance of a class C' inherited from class C; with versioning, an FE would not be allowed to contain an LFB instance for more than one version of a particular class.

LFB class versioning is supported by requiring a version string in the class definition. CEs may support backwards compatibility between multiple versions of a particular LFB class, but FEs are not allowed to support more than one single version of a particular class.

3.2.6. LFB Inheritance

LFB class inheritance is supported in the FE model as a means of defining new LFB classes. This also allows FE vendors to add vendor-specific extensions to standardized LFBs. An LFB class specification MUST specify the base class (with version number) it inherits from (with the default being the base LFB class). Multiple-inheritance is not allowed, though, to avoid the unnecessary complexity.

Inheritance should be used only when there is significant reuse of the base LFB class definition. A separate LFB class should be defined if there is not enough reuse between the derived and the base LFB class.

An interesting issue related to class inheritance is backward compatibility (between a descendant and an ancestor class). Consider the following hypothetical scenario where there exists a standardized LFB class "L1". Vendor A builds an FE that implements LFB "L1" and vendors B builds a CE that can recognize and operate on LFB "L1". Suppose that a new LFB class, "L2", is defined based on the existing "L1" class (for example, by extending its capabilities in some incremental way). Lets first examine the FE backward compatibility issue by considering what would happen if vendor B upgrades its FE from "L1" to "L2" while vendor C's CE is not changed. The old L1-based CE can interoperate with the new L2-

based FE if the derived LFB class "L2" is indeed backward compatible with the base class "L1".

The reverse scenario is a much less problematic case, i.e., when CE vendor B upgrades to the new LFB class "L2", but the FE is not upgraded. Note that as long as the CE is capable of working with older LFB classes, this problem does not affect the model; hence we will use the term "backward compatibility" to refer to the first scenario concerning FE backward compatibility.

Inheritance can be designed into the model with backward compatibility support by constraining the LFB inheritance such that the derived class is always a functional superset of the base class, i.e., the derived class can only grow on top of the base class, but not shrink from it. Additionally, the following mechanisms are required to support FE backward compatibility:

1) When detecting an LFB instance of an LFB type that is unknown to the CE, the CE MUST be able to query the base class of such an LFB from the FE.
2) The LFB instance on the FE SHOULD support a backward compatibility mode (meaning the LFB instance reverts itself back to the base class instance), and the CE SHOULD be able to configure the LFB to run in such mode.

3.3. FE Datapath Modeling

Packets coming into the FE from ingress ports generally flow through multiple LFBs before leaving out of the egress ports. How an FE treats a packet depends on many factors, such as type of the packet (e.g., IPv4, IPv6 or MPLS), actual header values, time of arrival, etc. The result of the operation of an LFB may have an impact on how the packet is to be treated in further (downstream) LFBs and this differentiation of packet treatment downstream can be conceptualized as having alternative datapaths in the FE. For example, the result of a 6-tuple classification (performed by a classifier LFB) controls what rate meter is applied to the packet (by a rate meter LFB) in a later stage in the datapath.

LFB topology is a directed graph representation of the logical datapaths within an FE, with the nodes representing the LFB instances and the directed link the packet flow direction from one LFB to the next. Section 3.3.1 discusses how the FE datapaths can be modeled as LFB topology; while Section 3.3.2 focuses on issues around LFB topology reconfiguration.

3.3.1. Alternative Approaches for Modeling FE Datapaths

There are two basic ways to express the differentiation in packet treatment within an FE, one representing the datapath directly and graphically (topological approach) and the other utilizing metadata (the encoded state approach).

. Topological Approach

Using this approach, differential packet treatment is expressed via actually splitting the LFB topology into alternative paths. In other words, if the result of an LFB must control how the packet is further processed, then such an LFB will have separate output ports (one for each alternative treatment) connected to separate sub-graphs (each expressing the respective treatment downstream).

. Encoded State Approach

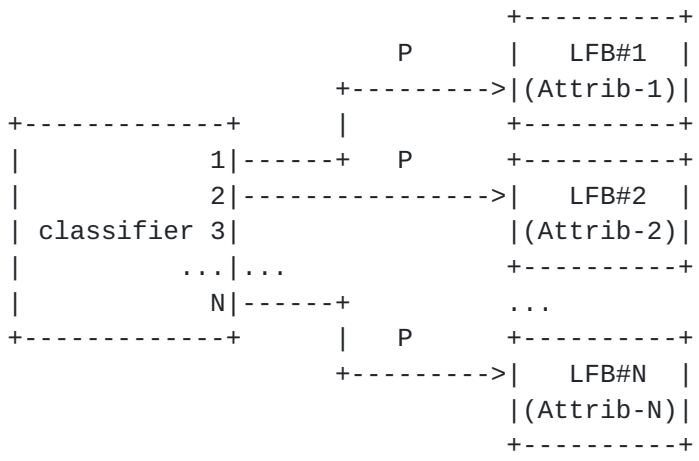
An alternative way of expressing differential treatment is using metadata. The result of the operation of an LFB can be encoded in a metadata which is passed along with the packet to downstream LFBs. A downstream LFB, in turn, can use the metadata (and its value, e.g., as an index into some table) to decide how to treat the packet.

Theoretically, the two approaches can substitute for each other, so one may consider using purely one (or the other) approach to describe all datapaths in an FE. However, neither model by itself is very useful for practically relevant cases. For a given FE with certain logical datapaths, applying the two different modeling approaches would result in very different looking LFB topology graphs. A model using purely the topological approach may require a very large graph with many links (i.e., paths) and nodes (i.e., LFB instances) to express all alternative datapaths. On the other hand, a model using purely the encoded state model would be restricted to a string of LFBs, which would make it very unintuitive to describe very different datapaths (such as MPLS and IPv4). Therefore, a mix of these two approaches will likely be used for a practical model. In fact, as we illustrate it below, the two approaches can be mixed even within the same LFB.

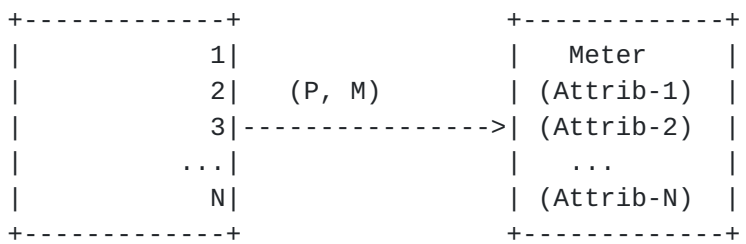
Using a simple example of a classifier with N classification outputs followed by some other LFBs, Figure 5(a) shows what the LFB topology looks like by using the purely topological approach. Each output from the classifier goes to one of the N LFBs followed and no metadata is needed here. The topological approach is simple, straightforward and graphically intuitive. However, if N is large and the N nodes followed the classifier (LFB#1, LFB#2, ..., LFB#N)

all belong to the same LFB type (for example, meter) but each with its own independent attributes, the encoded state approach gives a much simpler topology representation, as shown in Figure 5(b). The encoded state approach requires that a table of N rows of meter attributes is provided in the Meter node itself, with each row representing the attributes for one meter instance. A metadata M is also needed to pass along with the packet P from the classifier to the meter, so that the meter can use M as a look-up key (index) to find the corresponding row of the attributes that should be used for any particular packet P.

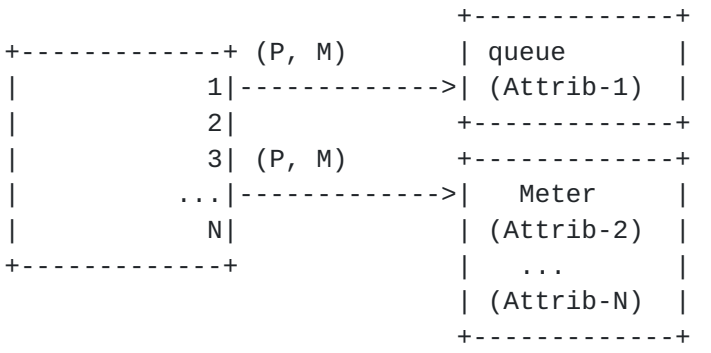
Now what if all the N nodes (LFB#1, LFB#2, ..., LFB#N) are not of the same type? For example, if LFB#1 is a queue while the rest are all meters, what is the best way to represent such datapaths? While it is still possible to use either the pure topological approach or the pure encoded state approach, the natural combination of the two seems the best by representing the two different functional datapaths using topological approach while leaving the N-1 meter instances distinguished by metadata only, as shown in Figure 5(c).



5(a) Using pure topological approach



5(b) Using pure encoded state approach to represent the LFB topology in 5(a), if LFB#1, LFB#2, ..., and LFB#N are of the same type (e.g., meter).

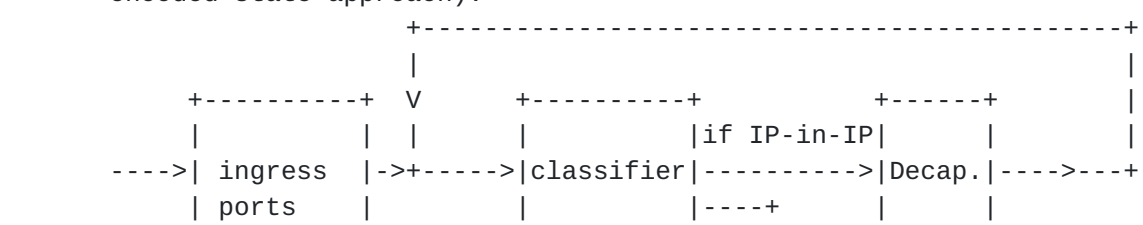


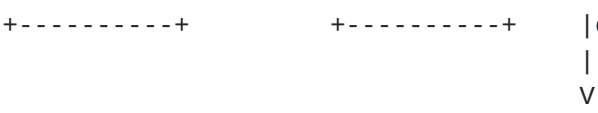
5(c) Using a combination of the two, if LFB#1, LFB#2, ..., and LFB#N are of different types (e.g., queue and meter).

Figure 5. An example of how to model FE datapaths

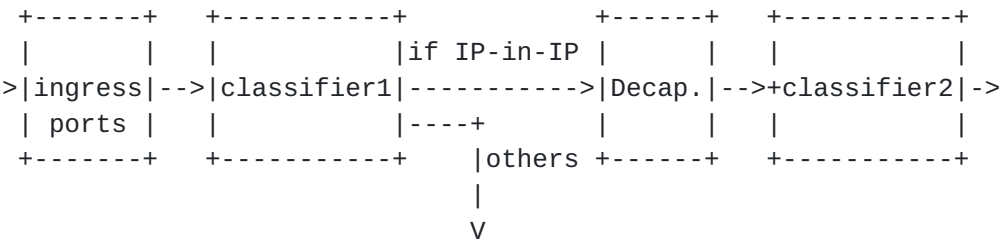
From this example, we demonstrate that each approach has distinct advantage for different situations. Using the encoded state approach, fewer connections are typically needed between a fan-out node and its next LFB instances of the same type, because each packet carries metadata with it so that the following nodes can interpret and hence invoke a different packet treatment. For those cases, a pure topological approach forces one to build elaborate graphs with a lot more connections and often results in an unwieldy graph. On the other hand, a topological approach is intuitive and most useful for representing functionally very different datapaths.

For complex topologies, a combination of the two is the most useful and flexible. Here we provide a general design guideline as to what approach is best used for what situation. The topological approach should primarily be used when the packet datapath forks into areas with distinct LFB classes (not just distinct parameterizations of the same LFB classes), and when the fan-outs do not require changes (adding/removing LFB outputs) at all or require only very infrequent changes. Configuration information that needs to change frequently should preferably be expressed by the internal attributes of one or more LFBs (and hence using the encoded state approach).





(a) The LFB topology with a logical loop



(b) The LFB topology without the loop utilizing two independent classifier instances.

Figure 6. An LFB topology example.

It is important to point out that the LFB topology here is the logical topology that the packets flow through, not the physical topology as determined by how the FE hardware is laid out. Nevertheless, the actual implementation may still influence how the functionality should be mapped into the LFB topology. Figure 6 shows one simple FE example. In this example, an IP-in-IP packet from an IPSec application like VPN may go to the classifier first and have the classification done based on the outer IP header; upon being classified as an IP-in-IP packet, the packet is then sent to a decapsulator to strip off the outer IP header, followed by a classifier again to perform classification on the inner IP header. If the same classifier hardware or software is used for both outer and inner IP header classification with the same set of filtering rules, a logical loop is naturally present in the LFB topology, as shown in Figure 6(a). However, if the classification is implemented by two different pieces of hardware or software with different filters (i.e., one set of filters for outer IP header while another set for inner IP header), then it is more natural to model them as two different instances of classifier LFB, as shown in Figure 6(b).

To distinguish multiple instances of the same LFB class, each LFB instance has its own LFB instance ID. One way to encode the LFB instance ID is to encode it as x.y where x is the LFB class ID while y is the instance ID within each LFB class.

3.3.2. Configuring the LFB Topology

While there is little doubt that the individual LFB must be configurable, the configurability question is more complicated for LFB topology. Since LFB topology is really the graphic representation of the datapaths within FE, configuring the LFB topology means dynamically changing the datapaths including changes to the LFBs along the datapaths on an FE, e.g., creating (i.e., instantiating) or deleting LFBs, setting up or deleting interconnections between outputs of upstream LFBs to inputs of downstream LFBs.

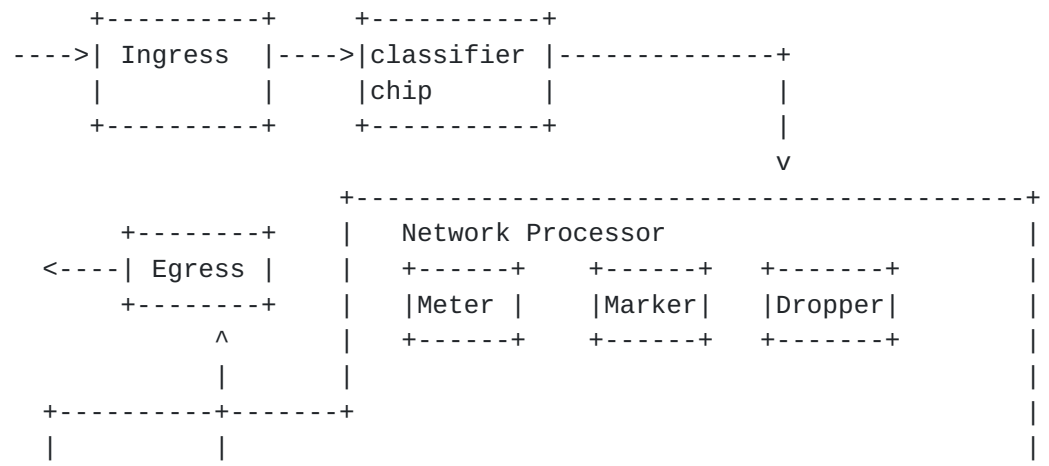
Why would the datapaths on an FE ever change dynamically? The datapaths on an FE is set up by the CE to provide certain data plane services (e.g., DiffServ, VPN, etc.) to the NE's customers. The purpose of reconfiguring the datapaths is to enable the CE to customize the services the NE is delivering at run time. The CE needs to change the datapaths when the service requirements change, e.g., when adding a new customer, or when an existing customer changes their service. However, note that not all datapath changes result in changes in the LFB topology graph, and that is determined by the approach we use to map the datapaths into LFB topology. As discussed in 3.3.1, the topological approach and encoded state approach can result in very different looking LFB topologies for the same datapaths. In general, an LFB topology based on a pure topological approach is likely to experience more frequent topology reconfiguration than one based on an encoded state approach. However, even an LFB topology based entirely on an encoded state approach may have to change the topology at times, for example, to totally bypass some LFBs or insert new LFBs. Since a mix of these two approaches is used to model the datapaths, LFB topology reconfiguration is considered an important aspect of the FE model.

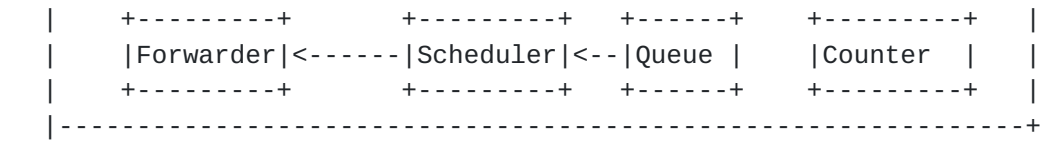
We want to point out that allowing a configurable LFB topology in the FE model does not mandate that all FEs must have such capability. Even if an FE supports configurable LFB topology, it is expected that there will be FE-specific limitations on what can actually be configured. Performance-optimized hardware implementation may have zero or very limited configurability, while FE implementations running on network processors may provide more flexibility and configurability. It is entirely up to the FE designers to decide whether or not the FE actually implements such reconfiguration and how much. Whether it is a simple runtime switch to enable or disable (i.e., bypass) certain LFBs, or more flexible software reconfiguration is all implementation detail internal to the FE but outside of the scope of FE model. In either case, the CE(s) must be able to learn the FE's configuration capabilities. Therefore, the FE model must provide a mechanism for

describing the LFB topology configuration capabilities of an FE. These capabilities may include (see Section 6 for details):

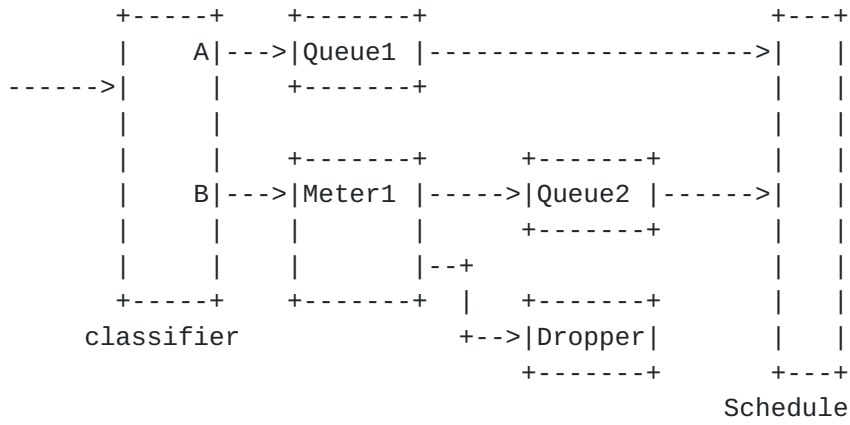
- . What LFB classes can the FE instantiate?
- . How many instances of the same LFB class can be created?
- . What are the topological limitations? For example:
  - o How many instances of the same class or any class can be created on any given branch of the graph?
  - o Ordering restrictions on LFBs (e.g., any instance of LFB class A must be always downstream of any instance of LFB class B).

Even if the CE is allowed to configure LFB topology for an FE, how can the CE interpret an arbitrary LFB topology (presented to the CE by the FE) and know what to do with it? In another word, how does the CE know the mapping between an LFB topology and a particular NE service or application (e.g., VPN, DiffServ, etc.)? We argue that first of all, it is unlikely that an FE can support any arbitrary LFB topology; secondly, once the CE understands the coarse capability of an FE, it is up to the CE to configure the LFB topology according to the network service the NE is supposed to provide. So the more important mapping that the CE has to understand is from the high level NE service to a specific LFB topology, not the other way around. Do we expect the CE has the ultimate intelligence to translate any high level service policy into the configuration data for the FEs? No, but it is conceivable that within a given network service domain (like DiffServ), a certain amount of intelligence can be programmed into the CE such that the CE has a general understanding of the LFBs involved and so the translation from a high level service policy to the low level FE configuration can be done automatically. In any event, this is considered an implementation issue internal to the control plane and outside the scope of the FE model. Therefore, it is not discussed any further in this draft.

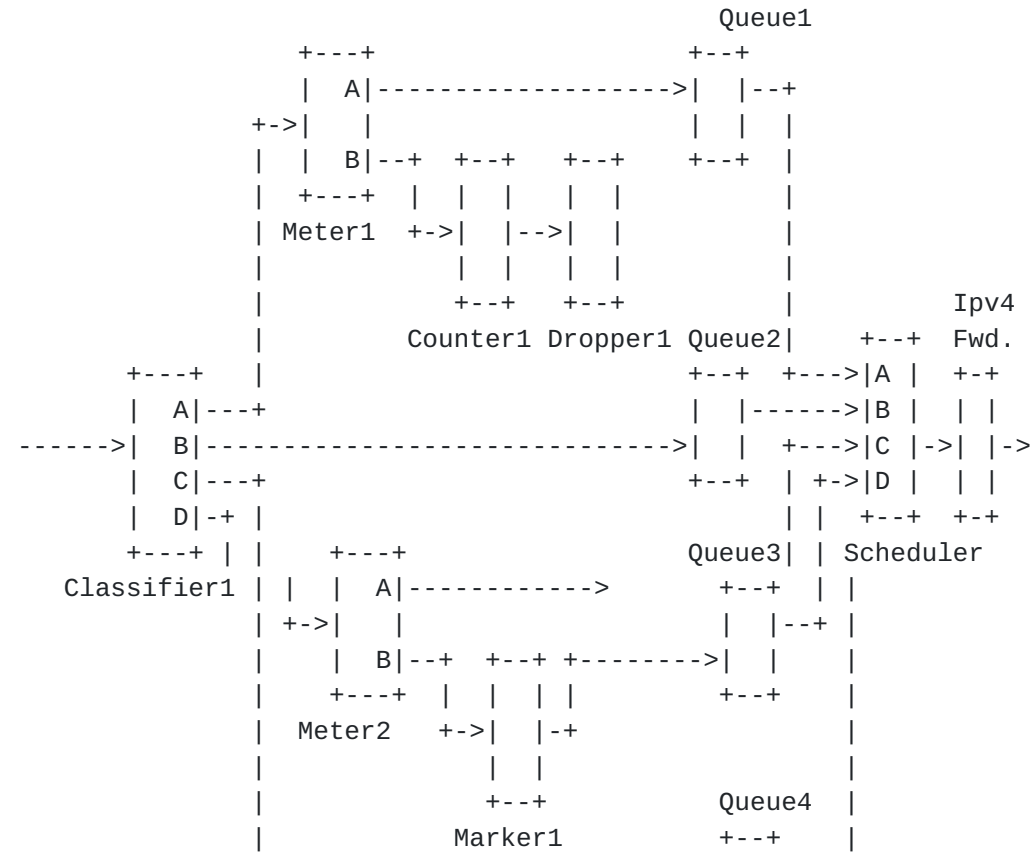




(a) The Capability of the FE, reported to the CE



(b) One LFB topology as configured by the CE and accepted by the FE



(c) Another LFB topology as configured by the CE and accepted by the FE

Figure 7. An example of configuring LFB topology.

Figure 7 shows an example where a QoS-enabled router has several line cards that have a few ingress ports and egress ports, a specialized classification chip, a network processor containing codes for FE blocks like meter, marker, dropper, counter, queue, scheduler and Ipv4 forwarder. Some of the LFB topology is already fixed and has to remain static due to the physical layout of the line cards. For example, all the ingress ports might be already hard wired into the classification chip and so all packets must follow from the ingress port into the classification engine. On the other hand, the LFBs on the network processor and their execution order are programmable, even though there might exist certain capacity limits and linkage constraints between these LFBs. Examples of the capacity limits might be: there can be no more than 8 meters; there can be no more than 16 queues in one FE; the scheduler can handle at most up to 16 queues; etc. The linkage constraints might dictate that classification engine may be followed by a meter, marker, dropper, counter, queue or IPv4 forwarder, but not scheduler; queues can only be followed by a scheduler; a scheduler must be followed by the IPv4 forwarder; the last LFB in the datapath before going into the egress ports must be the IPv4 forwarder, etc.

Once the FE reports such capability and capacity to the CE, it is now up to the CE to translate the QoS policy into the desirable configuration for the FE. Figure 7(a) depicts the FE capability while 7(b) and 7(c) depict two different topologies that the FE might be asked to configure to. Note that both the ingress and egress are omitted in (b) and (c) for simple representation. The topology in 7(c) is considerably more complex than 7(b) but both are feasible within the FE capabilities, and so the FE should accept either configuration request from the CE.

4. LFB Model -- LFB and Associated Data Definitions

The main goal of the FE model is to provide an abstract, generic, modular, implementation independent representation of the FEs. This is facilitated using the concept of LFBs which are instantiated from LFB classes. The LFB model is defined in this section to

describe the content and structures in LFB and associated data type definition.

The core part of the model is the definition of LFB classes. Section 4.4 provides more discussion on what will be part of an LFB class definition.

Operational parameters of the LFBs that must be visible to the CEs are conceptualized in the model as the LFB attributes. These include, for example, flags, single parameter arguments, complex arguments, and tables. The definition of the attributes of an LFB MUST be part of the LFB class definition. To promote consistent and terse definitions of the attributes of LFB classes, commonly used attribute types SHOULD be defined in the model outside of the LFB class definitions, so that LFB class definitions can "share" these type definitions by simply referring to the types. What will comprise a data type definition is further discussed in Section 4.1.

LFBs form a directed graph with each other by sending and receiving packets and associated metadata. To provide consistency and logical inter-operability among LFB classes, packet types (generic frame types) and metadata types MUST BE specified outside of the LFB class definitions (but part of the LFB model), so that the LFB class definitions can simply refer to these types. These blocks are further discussed in Section 4.3 and Section 4.2, respectively. In summary, the LFB model will consist of the following four categories of definitions:

1) Common data type definitions (Section 4.1)
2) Metadata definitions (Section 4.2);
3) Frame format definitions (Section 4.3);
4) LFB class definitions (Section 4.4).

It is not expected that the above information is exchanged between FEs and CEs "over-the-wire". But the model will serve as an important reference for the design and development of the CEs (software) and FEs (mostly the software part).

4.1. General Data Type Definitions

Data types will be used to describe the LFB attributes (see Section 4.4.4). This is similar to the concept of having a common header file for shared data types. Data types will include atomic data types (e.g. integer, ASCII string), as well as compound or derived data types (such as arrays and structures). Given that the FORCES protocol will be getting and setting attribute values, all atomic data types used here must be able to be conveyed in the FORCES

protocol. Further, the FORCES protocol will need a mechanism to convey compound data types. Details of such representation are for the protocol document, not the model documents.

Compound data types can build on atomic data types and other compound data types. There are three ways that compound data types can be defined. They may be defined as an array of elements of some compound or atomic data type. They may be a structure of named elements of compound or atomic data types (ala C structures). They may also be defined as augmentations (explained below in 4.1.3) of existing compound data types.

In addition, any data type may be used to define a new type by restricting the range of values that an instance of the data type can take on, and specifying specific semantics that go with that. This is similar to the SNMP notion of a textual convention.

For each data type the following information MUST be provided:
. Symbolic name of data type. Example: "T_IPV4ADDRESS".
. Actual type declaration.

In addition, a data type definition MAY include the following:
- . Range restrictions.
- . A set of symbolic names for special values. Example: "IPV4ADDR_LOOPBACK".

Note that not all attributes will exist at all times in all implementations. While the capabilities will frequently indicate this non-existence, CEs may attempt to reference non-existent or non-permitted attributes anyway. The FORCES protocol mechanisms should include appropriate error indicators for this case.

4.1.1. Arrays

Compound data types can be defined as arrays of compound or atomic data types. Arrays can only be subscripted by integers, and will be presumed to start with subscript 0. The mechanism defined above for non-supported attributes can also apply to attempts to reference non-existent array elements or to set non-permitted elements. The valid range of the subscripts of the array must be defined either in the definition of the array or in the LFB class which uses the compound type definition.

4.1.2. Structures

A structure is comprised of a collection of data elements. Each data element has a data type (either an atomic type or an existing

compound type.) and is assigned a name unique within the scope of the compound data type being defined. These serve the same function as "struct" in C, etc.

4.1.3. Augmentations

Compound types can also be defined as augmentations of existing compound types. If the existing compound type is a structure, augmentation may add new elements to the type. They may replace the type of an existing element with an augmentation derived from the current type. They may not delete an existing element, nor may they replace the type of an existing element with one that is not an augmentation of the type that the element has in the basis for the augmentation. If the existing compound type is an array, augmentation means augmentation of the array element type.

One consequence of this is that augmentations are compatible with the compound type from which they are derived. As such, augmentations are useful in defining attributes for LFB subclasses with backward compatibility. In addition to adding new attributes to a class, the data type of an existing attribute may be replaced by an augmentation of that attribute, and still meet the compatibility rules for subclasses.

For example, consider a simple base LFB class A that has only one attribute (attr1) of type X. One way to derive class A1 from A can be simply adding a second attribute (of any type). Another way to derive a class A2 from A can be replacing the original attribute (attr1) in A of type X with one of type Y, where Y is an augmentation of X. Both classes A1 and A2 are backward compatible with class A.

4.2. Metadata Definitions

For each metadata type, the following MUST be specified:

- Metadata symbolic name. Used to refer to the metadata type in LFB type specifications. Example: META_CLASSID.
- Brief synopsis of the metadata. Example: "Result of classification (0 means no match)".
- Data type and valid range.

In addition, the following information MAY BE part of the metadata definition:

- Symbolic definitions for frequently used or special values of the metadata.

4.3. Frame Format Definitions

This part of the LFB model will list packet types (frame types in general) that LFB classes can receive at their inputs and/or emit at their outputs.

For each distinct frame type, the following MUST be provided:
. Symbolic name of frame type. Example: FRAME_IPV4.
. Brief synopsis of the frame type. Example: "IPv4 packet".

4.4. LFB Class Definitions

Each LFB Class definition must provide the following information:
. Symbolic name of LFB class. Example: "LFB_IPV4_LPM"
. Short synopsis of LFB class. Example: "IPv4 LPM Lookup LFB"
. Version indicator
. Inheritance indicator (see discussion in Section 4.4.1)
. Inputs (see discussion in Section 4.4.2)
. Outputs (see discussion in Section 4.4.3)
. Attributes (see discussion in Section 4.4.4)
. Operational specification (see discussion in Section 4.4.5)

4.4.1. LFB Inheritance

To support LFB class inheritance, the LFB specification must have a place holder for indicating the base class and its version. It is assumed that the derived class is backward compatible with the base class.

4.4.2. LFB Inputs

An LFB class may have zero, one, or more inputs. We assume that most LFBs will have exactly one input. Multiple inputs with the same input type are modeled as one input group. The input group should count as one entry in the input specification. The number of inputs (including input groups) is fixed.

Multiple inputs with different input type should be avoided if possible (see discussion in Section 3.2.1). Some special LFBs will have no inputs at all. For example, a packet generator LFB does not need an input.

The LFB class definition MUST specify whether or not the number of inputs of the LFB is fixed, and the exact number if fixed. For each LFB input (group), the following MUST be specified:

- Symbolic name of input. Example: "PKT_IN". Note that this symbolic name must be unique only within the scope of the LFB class.
- Brief synopsis of the input. Example: "Normal packet input".
- Indication of whether this input is an input group (i.e., if it is allowed to be instantiated).
- List of allowed frame formats. Example: "{FRAME_IPV4, FRAME_IPV6}". Note that this list should refer to symbols specified in the frame definition of the LFB model (see Section 4.3).
- List of required metadata. Example: {META_CLASSID, META_IFID}. This list should refer to symbols specified in the metadata definition of the LFB model (see Section 4.2). For each metadata it should be specified whether the metadata is required or optional. For each optional metadata a default value MAY BE specified, which is used by the LFB if the metadata is not provided at the input.

4.4.3. LFB Outputs

An LFB class may have zero, one, or more outputs. If there are multiple outputs with the same output type, we model them as output group. Some special LFBs may have no outputs at all (e.g., Dropper).

The number of outputs may be fixed for some LFB types and may be configurable for others. The LFB Class definition MUST specify the number of outputs (or output types) of the LFB. The output group should count as one entry in the output specification, but the entry should indicate that instantiation of the output is allowed.

For each LFB output (group) the following MUST be specified:

- Symbolic name of the output. Example: "UNPROC". In case of an output group, the symbolic name is the prefix used to construct unique symbols for each output instance. Example: "PKTOUT". Note that the symbolic name must be unique only within the scope of the LFB class.
- Brief synopsis of the output. Example: "Normal packet output".
- Indication of whether this output is an output group (i.e., if it is allowed to be instantiated).
- List of allowed frame formats. Example: "{FRAME_IPV4, FRAME_IPV6}". Note that this list should refer to symbols specified in the frame definition of the LFB model (see Section 4.3).
- List of emitted (generated) metadata. Example: {META_CLASSID, META_IFID}. This list should refer to symbols specified in the

metadata definition of the LFB model (see Section 4.2). For each generated metadata, it should be specified whether the metadata is always generated or generated only in certain conditions. This information is important when assessing compatibility between LFBs.

4.4.4. LFB Attributes

The operational state of the LFB is modeled by the variables of the LFB, collectively called attributes. Note that the attributes here refer to the operational parameters of the LFBs that must be visible to the CEs. The other variables that are internal to LFB implementation are not included here in the LFB attributes and are not modeled here.

Attribute types will include the following three categories:

. Capability attributes (see Section 9.4 for more on LFB capabilities). Examples:
  * Supported optional features of the LFB class;
  * Maximum number of configurable outputs for an output group;
  * Metadata pass-through limitations of the LFB;
  * Maximum size of configurable attribute tables;
  * Supported access modes of certain attributes (see below).
. Operational attributes, some of them are configurable by the CE, while others might be internally maintained state which are read-only for the CE and necessary for the CE to operate properly. Examples:
  * Configurable flags and switches selecting between operational modes of the LFB;
  * ARP tables;
  * Number of outputs in an output group;
  * Metadata CONSUME vs. PROPAGATE mode selector.
. Statistical attributes (collected by the FE and provided for reading to the CE). Examples:
  * Packet and byte counters;
  * Other event counters.

Some of the attributes will be generically available in all LFBs while others will be specific to the LFB class. Examples of generic LFB attributes are:

. LFB class inheritance information (see Section 4.4.1)
. Number and type of inputs (in case the LFB is self-descriptive)
. Number and type of outputs (in case the LFB is self-descriptive)
. Number of current outputs for each output group
. Metadata CONSUME/PROPAGATE mode selector

There may be various access permission restrictions on what the CE can do with an LFB attribute. The following categories may be supported:

- No-access attributes. This is useful when multiple access modes maybe defined for a given attribute to allow some flexibility for different implementations.
- Read-only attributes.
- Read-write attributes.
- Write-only attributes. This could be any configurable data for which read capability is not provided to the CEs. (??? Do we have good example???)
- Read-reset attributes. The CE can read and reset this resource, but cannot set it to an arbitrary value. Example: Counters.
- Firing-only attributes. A write attempt to this resource will trigger some specific actions in the LFB, but the actual value written is ignored. (??? Example???)

The LFB class may define more than one possible access mode for a given attribute (for example, write-only and read-write), in which case it is left to the actual implementation to pick one of the modes. In such cases a corresponding capability parameter must inform the CE of which mode the actual LFB instance supports. The attributes of the LFB class must be defined as a list. For each attribute the following information MUST be provided:

- Reference to the data type (e.g., specified in the generic data type block of the LFB model or in an LFB specific data type block).
- Access permission(s).
- Additional range restrictions (i.e., beyond what is specified by the data type definition).
- Default value. Applied when the LFB is initialized or reset.

The actual structuring of LFB attributes requires further study.

4.4.5. LFB Operational Specification

This section of the model should verbally describe what the LFB does. This will most likely be embedded in an unstructured text field in the model.

5. LFB Topology Model (To be written)

(Editor's note: This is a place holder to describe the details on how to model LFB topology.)

6. FE Level Attributes (To be written)

(Editor's note: This is a place holder to describe the FE level attributes including FE capabilities, for examples:

- . How this FE is connected with other FEs (if known by the FE)?
- . What LFB classes can the FE instantiate?
- . How many instances of the same LFB class can be created?
- . What are the topological limitations? For example:
  - o How many instances of the same class or any class can be created on any given branch of the graph?
  - o Ordering restrictions on LFBs (e.g., any instance of LFB class A must be always downstream of any instance of LFB class B).

)

7. LFB Class Library

A set of LFB classes are identified here in the LFB class library as necessary to build common FE functions.

Several working groups in the IETF have already done some relevant work in modeling the provisioning policy data for some of the functions we are interested in, for example, DiffServ (Differentiated Services) PIB [4], IPSec PIB [8]. Whenever possible, we should try to reuse the work done elsewhere instead of reinventing the wheel.

7.1. Port LFB

A Port LFB is used to map a physical port into the LFB model.

The Port LFB maps sources and sinks of packets from outside the LFB model onto one logical block which defines and models a physical port implementing those functions.

The Port LFB contains a number of configurable parameters, which may include, but are not limited to, the following items:

- . the number of ports on this LFB;
- . the sub-interfaces if any;
- . the static attributes of each port (e.g., port type, direction, link speed);
- . the configurable attributes of each port (e.g., IP address, administrative status);
- . the statistics collected on each port (e.g., number of packets received);

. the current status (up or down).

The Port LFB can have three modes of operation:
. ingress only
. egress only
. hybrid (contains ingress and egress functions)

7.2. Dropper LFB

A dropper LFB has one input, and no outputs. It discards all packets that it receives without any modification or examination of those packets.

The purpose of a dropper LFB is to allow the description of "sinks" within the model, where those sinks do not result in the packet being sent into any object external to the model.

7.3. Redirector (de-MUX) LFB

A redirector LFB has one input, and N outputs.

The purpose of the redirector LFB is to explicitly represent a place in the LFB Topology where the redirection process occurs, and where it may be configured.

The redirector LFB takes an input packet P, and uses the metadata M to redirect that packet to one or more of N outputs, e.g. unicast forwarding, multicast, or broadcast.

Note that other LFBs may also have redirecting functionality, if they have multiple outputs.

7.4. Scheduler LFB

A Scheduler LFB has multiple inputs and one output. The purpose of the Scheduler LFB is to perform time-dependent packet forwarding. The Scheduler LFB multiplexes from its inputs onto its output(s), based on internal configuration such as packet priority, etc. The packet is not modified during this process.

7.5. Queue LFB

The Queue LFB has one input and one output. It takes input packets and places them onto queues. These packets are later forwarded to the output(s) of the LFB, based on back-pressure from the next LFB which typically is a scheduler LFB.

7.6. Counter LFB

A counter LFB updates its statistical attributes, by counting packets, or metadata. The packet is not modified, and the metadata may, or may not, be modified.

The purpose of a Counter LFB is to record simple accounting of events on the FE.

A counter LFB is independent of time 't', in that it does not perform any time-dependent counting. The time at which a count is made may, however, be associated with that count.

7.7. Meter LFB and Policer LFB

A Meter LFB is a counter LFB that is time dependent. That is, it meters the rate over time at which packets or metadata flow through the LFB. The purpose of the Meter LFB is to record time-dependent accounting of events on the FE.

When a Meter LFB has multiple outputs, with one output being a marker, or dropping the packet, then the Meter LFB becomes a Policer LFB, performing a policing function.

7.8. Classifier LFB

A Classifier LFB uses its attributes to classify the packet into one of N different logical classes.

The purpose of a Classifier LFB is to logically partition packets into one or more classes. The result of this partitioning is that the Classifier LFB produces metadata that describes the classes into which the packet has been partitioned. The packet is not modified during this process.

A Classifier LFB takes an input packet and metadata, and produces the same packet with new or more metadata. A classifier is parameterized by filters. Classification is done by matching the contents of the incoming packets according to the filters, and the result of classification is produced in the form of metadata. Note that this classifier is modeled solely based on its internal processing, and not on its inputs and outputs. The block is a single-exit classifier that does NOT physically redirect the packet. In contrast, a DiffServ-like classifier is a 1:N (fan-out) device: It takes a single traffic stream as input and generate N logically separate traffic streams as output. That kind of multi-

exit classifier can be modeled by combining this classifier with a redirector (see Section 6.1.6).

A filter decides if input packets match particular criteria. That is, it "marks" a packet as either matching, or non-matching to the filter criteria. According to [DiffServ], "a filter consists of a set of conditions on the component values of a packet's classification key (the header values, contents, and attributes relevant for classification)".

Note that other FE LFBs MAY perform simple classification on the packet or metadata. The purpose of the FE Classifier LFB is to model an LFB that "digests" large amounts of input data (packet, metadata), to produce a "summary" of the classification results, in the form of additional metadata. Other FE LFBs can then use this summary information to quickly and simply perform trivial "classifications".

The requirement for a unique and separate FE Classifier LFB comes about because it would not make sense to model a classifier LFB inside each of every other LFB. Such a model would be highly redundant. We therefore specifically model a complex classification LFB, and explicitly state that other blocks may make decisions based on the parameters S, t, and M, but not on P.

Note that a classifier LFB may have multiple outputs. In that case, it may redirect input packets to one (or more) of the outputs, and may not associate any metadata with those output packets.

7.9. Modifier LFB

A modifier LFB modifies incoming packets and sends them out. Usually the metadata is used to determine how to modify the packet.

This LFB is defined in a generic manner, and we expect that specific examples of packet and/or metadata modification will be described as a subclass of the modifier LFB.

For example, we may have an explicit LFB for packet compression and decompression, or for encryption and decryption, or for packet encapsulation. The decision as to how best to model these functions will be made based on further investigation of the LFB model, and with practical experience using it.

7.10. Packet Header Rewriter LFB

This LFB is used to re-write fields on the packet header, such as IPv4 TTL decrementing, checksum calculation, or TCP/IP NAT.

We may want to have multiple LFBs for different kinds of header re-writing.

8. Satisfying the Requirements on FE Model

(Editor's Note: The text in this section is very preliminary but we decide to leave it as is because it is too early to understand how to model all the functions as dictated in [1] when Section 7 is still very much work in progress. This section should be revised once Section 7 is more settled.)

A minimum set of FE functions is defined in [1] that must be supported by any proposed FE model. In this section, we demonstrate how the three components in FE model as described in Section 4, 5, 6 along with the LFB class library defined in Section 7 can be used to express all the logical functions required in [1].

8.1. Port Functions

Every FE contains a certain number of interfaces (ports), including both the inter-NE interfaces and intra-NE interfaces. The inter-NE interfaces are the external interfaces for the NE to receive/forward packets from/to the external world. The intra-NE interfaces are used for FE-FE or FE-CE communications. Same model should be used for both the inter-FE and intra-FE interfaces, but it is necessary to make the distinction between the two known to the CE so that the CE can do different configuration.

The port LFB class is designed to model the specific physical ports while the source/sink LFB can be used to model the logical interface.

The intra-NE interfaces that are used for FE-FE communications should be modeled just like the inter-NE interfaces. The ForCES base protocol will include FE topology query so that the CE can learn of how the multiple FEs are interconnected via such interfaces. But the intra-NE interfaces that are used for FE-CE communications are part of the ForCES protocol entity on the FE and so it is not necessary to model them explicitly. It is assumed that every FE will have at least one internal interface to communicate to the CE and such interface do not have to be visible in the FE model.

8.2. Forwarding Functions

Support for IPv4 and IPv6 unicast and multicast forwarding functions must be provided by the model.

Typically, the control plane maintains the Routing Information Base (RIB), which contains all the routes discovered by all the routing protocols with all kinds of attributes relevant to the routes. The forwarding plane uses a different database, the Forwarding Information Base (FIB), which contains only the active subset of those routes (only the best routes chosen for forwarding) with attributes that are only relevant for forwarding. A component in the control plane, termed Route Table Manager (RTM), is responsible to manage the RIB in the CE and maintain the FIB used by the FEs. Therefore, the most important aspect in modeling the forwarding functions is the data model for the FIB. The model also needs to support the possibility of multiple paths.

At the very minimum, each route in the FIB needs to contain the following layer-3 information:

- the prefix of the destination IP address;
- the length of the prefix;
- the number of equal-cost multi-path;
- the next hop IP address and the egress interface for each path.

Another aspect of the forwarding functions is the method to resolve a next hop destination IP address into the associated media address. There are many ways to resolve Layer 3 to Layer 2 address mapping depending upon link layer. For example, in case of Ethernet links, the Address Resolution Protocol (ARP, defined in RFC 826) is used for IPv4 address resolution.

Assuming a separate table is maintained in the FEs for address resolution, the following information is necessary for each address resolution entry:

- the next hop IP address;
- the media address.

Different implementation may have different ways to maintain the FIB and the resolution table. For example, a FIB may consist of two separate tables, one to match the prefix to the next hop and the other to match the next hop to the egress interface. Another implementation may use one table instead. Our approach of using the fine-grained FE blocks to model the forwarding functions allow such flexibility.

For example, a combination of a classifier, followed by a modifier and a redirector can model the forwarding function.

8.3. QoS Functions

The IETF community has already done lots work in modeling the QoS functions in the datapath. The IETF DiffServ working group has defined an informal data model [3]for QoS-related functions like classification, metering, marking, actions of marking, dropping, counting and multiplexing, queueing, etc. The latest work on DiffServ PIB (Policy Information Base) [4] defines a set of provisioning classes to provide policy control of resources implementing the Diferentiated Services Architecture. DiffServ PIB also has an element of capability flavor to it. The IETF Policy Framework working group is also defining an informational model [6] to describe the QoS mechanisms inherent in different network devices, including hosts. This model is intended to be used with the QoS Policy Information Model [7] to model how policies can be defined to manage and configure the QoS mechanisms present in the datapath of devices.

Here is a list of QoS functions that should be supported by the FE model:

- Classifier
- Meter
- Marker
- Dropper
- Counter
- Queue and Scheduler
- Shaper

LFB class library as described in Section 7 already supports most of these functions directly.

Note that A shaper should be modeled as a queue feeding a scheduler input that is serviced using a non-work-conserving policy. The queue LFB would include multiple FIFO queue resources (selected by META_QUEUE_ID) and AQManagers assigned to queues. The scheduler LFB would include multiple input resources with associated service policies. Queue outputs would be bound to scheduler inputs via passing META_SCHED_ID with the packet at the output of the queue. The metadata is only there to allow correlation in configuration parameters between the queueing LFB and the scheduler LFB (assign queue X to scheduler input Y by configuring queue X to emit META_SCHED_ID Y).

8.4. Generic Filtering Functions

A combination of classifier, redirector, modifier etc. can model complex set of filtering functions. For example, Figure 8 represents a filtering function that classifies packets into one of two logical classes: forward, and drop. These logical classes are represented as meta data M1, and M2. The re-director uses this meta data to re-direct the packet to one of two outputs. The first sinks the packet back into the network. The second silently drops the packets.

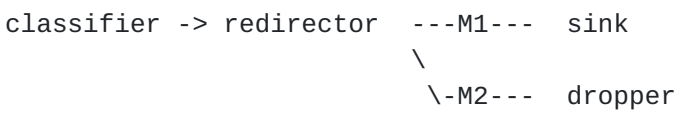


Figure 8. A filtering function example.

8.5. Vendor Specific Functions

New LFB class can always be defined according to the LFB model as described in Section 7 to support vendor specific functions. New LFB class can also be derived from an existing LFB class by inheritance.

8.6.High-Touch Functions

High-touch functions are those that take action on the contents or headers of a packet based on content other than what is found in the IP header. Examples of such functions include NAT, ALG, firewall, tunneling and L7 content recognition.

The ForCES working group first needs to agree upon a small set of common high-touch functions with well-defined behavior to be included in the LFB class library. Here is a list of candidate blocks:

- NAT
- Firewall
- Encapsulator
- Decapsulator

8.7. Security Functions

The FE model must be able to describe the types of encryption and/or decryption functions that an FE supports and the associated attributes for such functions.

The IP Security Policy (IPSP) Working Group in the IETF has started work in defining the IPSec Policy Information Base [8]. Further

study on this is needed to determine whether it can be reused here and any other additional work is needed.

8.8. Off-loaded Functions

In addition to the packet processing functions that are typical to find on the FEs, some logical functions may also be executed asynchronously by some FEs, according to a certain finite-state machine, triggered not only by packet events, but by timer events as well. Examples of such functions include finite-state machine execution required by TCP termination or OSPF Hello processing off-loaded from the CE. The FE model must be capable of expressing these asynchronous functions, so that the CE may take advantage of such off-loaded functions on the FEs.

The ForCES working group first needs to agree upon a small set of such off-loaded functions with well-understood behavior and interactions with the control plane.

8.9. IPFLOW/PSAMP Functions

[9] defines architecture for IP traffic flow monitoring, measuring and exporting. The LFB model supports statistics collection on the LFB by including statistical attributes (Section 4.4.4) for all the LFB class definitions, and meter LFB (Section 7.2.2) and counter LFB (Section 7.2.1) can also be used to support accounting functions in the FE.

[10] describes a framework to define a standard set of capabilities for network elements to sample subsets of packets by statistical and other methods. Time event generation, filter LFB, and counter/meter LFB are the elements needed to support packet filtering and sampling functions -- these elements are all included in the FE model.

9. Using the FE model in the ForCES Protocol

The actual model of the forwarding plane in a given NE is something the CE must learn and control via communicating with the FEs (or by other means). Most of this communication will happen in the post-association phase using the ForCES protocol. The following types of information must be exchanged between CEs and FEs via the ForCES protocol:

1) FE topology query;
2) FE capability declaration;
3) LFB topology (per FE) and configuration capabilities query;
4) LFB capability declaration;

5) State query of LFB attributes;
6) Manipulation of LFB attributes;
7) LFB topology reconfiguration.

Items 1) through 5) are query exchanges, the main flow of information being from the FEs to the CEs. Items 1) through 4) are typically queried by the CE(s) in the beginning of the post-association (PA) phase, though they may be repeatedly queried at any time in the PA phase. Item 5) (state query) will be used at the beginning of the PA phase, and often frequently during the PA phase (especially for the query of statistical counters).

Items 6) and 7) are "command" type of exchanges, the main flow of information being from the CEs to the FEs. Messages in Item 6) (the LFB re-configuration commands) are expected to be used frequently. Item 7) (LFB topology re-configuration) is needed only if dynamic LFB topologies are supported by the FEs and it is expected to be used infrequently.

Among the seven types of payload information the ForCES protocol carries between CEs and FEs, the FE model covers all of them except item 1), which concerns the inter-FE topology. The FE model focuses on the LFB and LFB topology within a single FE. Since the information of item 1) requires global knowledge about all the FEs and their inter-connection with each other, this exchange is made part of the ForCES base protocol instead of the FE model.

The relationship between the FE model and the seven post-association messages are visualized in Figure 9:

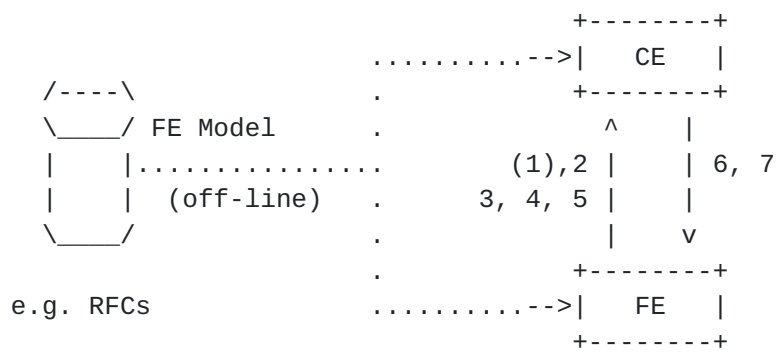


Figure 9. Relationship between FE model and the ForCES protocol messages, where (1) is part of the ForCES base protocol, and the rest are defined by the FE model.

The actual encoding of these messages is defined by the ForCES protocol and beyond the scope of the FE model. Their discussion is nevertheless important here for the following reasons:

- These PA model components have considerable impact on the FE model. For example, some of the above information can be represented as attributes of the LFBs, in which case such attributes must be defined in the LFB classes.
- The understanding of the type of information that must be exchanged between the FEs and CEs can help to select the appropriate protocol format and the actual encoding method (such as XML, TLVs).
- Understanding the frequency of these types of messages should influence the selection of the protocol format (efficiency considerations).

The remaining sub-sections of this section address each of the seven message types.

9.1. FE Topology Query

(Editor's Note: It is still an open issue where the FE topology information query belongs -- it can be either supported as part of FE attributes in the FE model, or it can be supported by the ForCES protocol explicitly. Hence the text here is tentative and subject to change per WG discussion.)

An FE may contain zero, one or more external ingress ports. Similarly, an FE may contain zero, one or more external egress ports. In another word, not every FE has to contain any external ingress or egress interfaces. For example, Figure 10 shows two cascading FEs. FE #1 contains one external ingress interface but no external egress interface, while FE #2 contains one external egress interface but no ingress interfce. It is possible to connect these two FEs together via their internal interfaces to achieve the complete ingress-to-egress packet processing function. This provides the flexibility to spread the functions across multiple FEs and interconnect them together later for certain applications.

While the inter-FE communication protocol is out of scope for ForCES, it is up to the CE to query and understand how multiple FEs are inter-connected to perform a complete ingress-egress packet processing function, like that described in Figure 10. The inter-FE topology information may be provided by FEs, may be hard-coded into CE, or may be provided by some other entity (e.g., a bus manager) independent of the FEs. So while the ForCES protocol supports FE topology query from FEs, it is optional for the CE to

use it, assuming the CE has other means to gather such topology information.

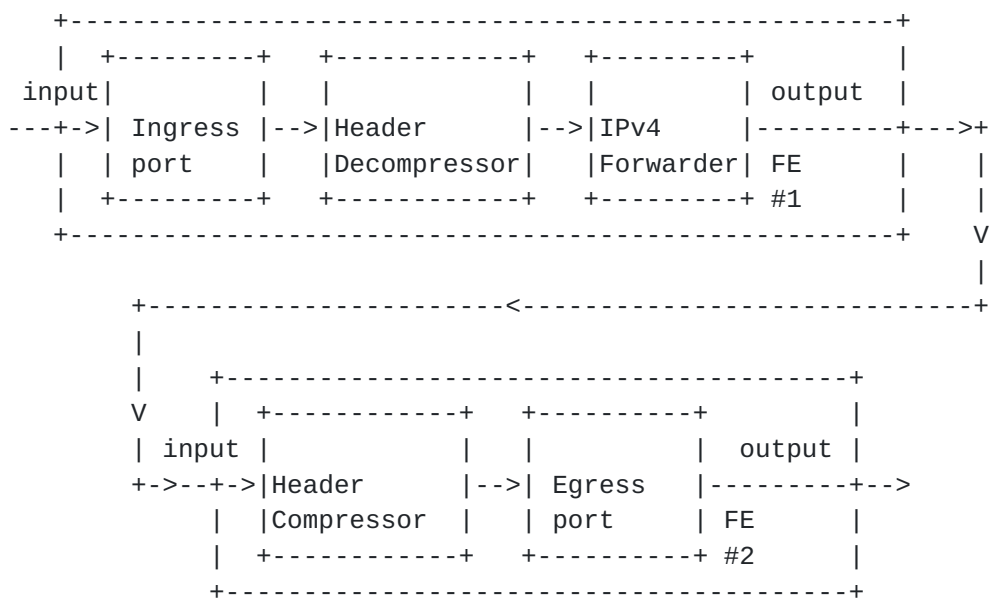


Figure 10. An example of two FEs connected together.

Once the inter-FE topology is discovered by the CE after this query, it is assumed that the inter-FE topology remains static. However, it is possible that an FE may go down during the NE operation, or a board may be inserted and a new FE activated, so the inter-FE topology will be affected. It is up to the ForCES protocol to provide mechanism for the CE to detect such events and deal with the change in FE topology. FE topology is outside the scope of the FE model.

9.2. FE Capability Declarations

FEs will have many types of limitations. Some of the limitations must be expressed to the CEs as part of the capability model. The CEs must be able to query these capabilities on a per-FE basis. Examples:

- Metadata passing capabilities of the FE. Understanding these capabilities will help the CE to evaluate the feasibility of LFB topologies, and hence to determine the availability of certain services.
- Global resource query limitations (applicable to all LFBs of the FE).
- LFB supported by the FE.
- LFB class instantiation limit.

. LFB topological limitations (linkage constraint, ordering etc.)

9.3. LFB Topology and Topology Configurability Query

The ForCES protocol must provide the means for the CEs to discover the current set of LFB instances in an FE and the interconnections between the LFBs within the FE. In addition, there should be sufficient information provided on whether the FE supports any CE-initiated (dynamic) changes to the LFB topology, and if so, what are the allowed topologies. Topology configurability can also be considered as part of the FE capability query as described in Section 9.3.

9.4. LFB Capability Declarations

LFB class specifications will define a generic set of capabilities. When an LFB instance is implemented (instantiated) on a vendor's FE, some additional limitations may be introduced. Note that we discuss here only limitations that are within the flexibility of the LFB class specification, that is, the LFB instance will remain compliant with the LFB class specification despite these limitations. For example, certain features of an LFB class may be optional, in which case it must be possible for the CE to determine if an optional feature is supported by a given LFB instance or not. Also, the LFB class definitions will probably contain very few quantitative limits (e.g., size of tables), since these limits are typically imposed by the implementation. Therefore, quantitative limitations should always be expressed by capability arguments.

LFB instances in the model of a particular FE implementation will possess limitations on the capabilities defined in the corresponding LFB class. The LFB class specifications must define a set of capability arguments, and the CE must be able to query the actual capabilities of the LFB instance via querying the value of such arguments. The capability query will typically happen when the LFB is first detected by the CE. Capabilities need not be re-queried in case of static limitations. In some cases, however, some capabilities may change in time (e.g., as a result of adding/removing other LFBs, or configuring certain attributes of some other LFB when the LFBs share physical resources), in which case additional mechanisms must be implemented to inform the CE about the changes.

The following two broad types of limitations will exist:

. Qualitative restrictions. For example, a standardized multi-field classifier LFB class may define a large number of

classification fields, but a given FE may support only a subset of those fields.
- Quantitative restrictions, such as the maximum size of tables, etc.

The capability parameters that can be queried on a given LFB class will be part of the LFB class specification. The capability parameters should be regarded as special attributes of the LFB. The actual values of these arguments may be, therefore, obtained using the same attribute query mechanisms as used for other LFB attributes.

Capability attributes will typically be read-only arguments, but in certain cases they may be configurable. For example, the size of a lookup table may be limited by the hardware (read-only), in other cases it may be configurable (read-write, within some hard limits).

Assuming that capabilities will not change frequently, the efficiency of the protocol/schema/encoding is of secondary concern.

9.5. State Query of LFB Attributes

This feature must be provided by all FEs. The ForCES protocol and the data schema/encoding conveyed by the protocol must together satisfy the following requirements to facilitate state query of the LFB attributes:

- Must permit FE selection. This is primarily to refer to a single FE, but referring to a group of (or all) FEs may optional be supported.
- Must permit LFB instance selection. This is primarily to refer to a single LFB instance of an FE, but optionally addressing of a group of LFBs (or all) may be supported.
- Must support addressing of individual attribute of an LFB.
- Must provide efficient encoding and decoding of the addressing info and the configured data.
- Must provide efficient data transmission of the attribute state over the wire (to minimize communication load on the CE-FE link).

9.6. LFB Attribute Manipulation

This is a place-holder for all operations that the CE will use to populate, manipulate, and delete attributes of the LFB instances on the FEs. This is how the CE configures an individual LFB instance.

The same set of requirements as described in Section 9.5 for attribute query applies here for attribute manipulation as well.

Support for various levels of feedback from the FE to the CE (e.g., request received, configuration completed), as well as multi-attribute configuration transactions with atomic commit and rollback, may be necessary in some circumstances.

(Editor's note: It remains an open issue as to whether or not other methods are needed in addition to "get attribute" and "set attribute" (such as multi-attribute transactions). If the answer to that question is yes, it is not clear whether such methods should be supported by the FE model itself or the ForCES protocol.)

9.7. LFB Topology Re-configuration

Operations that will be needed to reconfigure LFB topology:

- Create a new instance of a given LFB class on a given FE.
- Connect a given output of LFB x to the given input of LFB y.
- Disconnect: remove a link between a given output of an LFB and a given input of another LFB.
- Delete a given LFB (automatically removing all interconnects to/from the LFB).

10. Acknowledgments

The authors would also like to thank the following individuals for their invaluable technical input: David Putzolu, Hormuzd Khosravi, Eric Johnson, David Durham, Andrzej Matejko, T. Sridhar, Jamal Hadi Salim, Alex Audu, Gamil Cain.

11. Security Considerations

The FE model describes the representation and organization of data sets and attributes in the FEs. ForCES framework document [2] provides a comprehensive security analysis for the overall ForCES architecture. For example, the ForCES protocol entities must be authenticated per the ForCES requirements before they can access the information elements described in this document via ForCES. The access to the information contained in the FE model is accomplished via the ForCES protocol which will be defined in separate documents and so the security issues will be addressed there.

12. Normative References

[1] Khosravi, H. et al., "Requirements for Separation of IP Control and Forwarding", work in progress, July 2003, <draft-ietf-forces-requirements-10.txt>.

13. Informative References

[2] Yang, L. et al., "Forwarding and Control Element Separation (ForCES) Framework", work in progress, July 2003, <draft-ietf-forces-framework-07.txt>.

[3] Bernet, Y. et al., "An Informal Management Model for Diffserv Routers", May 2002.

[4] Chan, K. et al., "Differentiated Services Quality of Service Policy Information Base", March 2003.

[5] Sahita, R. et al., "Framework Policy Information Base", RFC 3318, March 2003.

[6] Moore, B. et al., "Information Model for Describing Network Device QoS Datapath Mechanisms", work in progress, May 2002, <draft-ietf-policy-qos-device-info-model-08.txt>.

[7] Snir, Y. et al., "Policy Framework QoS Information Model", work in progress, Nov 2001, <draft-ietf-policy-qos-info-model-04.txt".

[8] Li, M. et al., "IPsec Policy Information Base", work in progress, January 2003, <draft-ietf-ipsp-ipsecpib-07.txt>.

[9] Quittek, J. et Al., "Requirements for IP Flow Information Export", work in progress, June 2003, <draft-ietf-ipfix-reqs-10.txt>.

[10] Duffield, N., "A Framework for Passive Packet Measurement ", work in progress, June 2003, <draft-ietf-psamp-framework-03.txt>.

[11] Pras, A. and Schoenwaelder, J., FRC 3444 "On the Difference between Information Models and Data Models", January 2003.

14. Authors' Addresses

L. Lily Yang
Intel Labs
2111 NE 25th Avenue
Hillsboro, OR 97124, USA
Phone: +1 503 264 8813
Email: lily.l.yang@intel.com

Joel M. Halpern
Megisto Systems, Inc.
20251 Century Blvd.
Germantown, MD 20874-1162, USA
Phone: +1 301 444-1783
Email: jhalpern@megisto.com

Ram Gopal
Nokia Research Center
5, Wayside Road,
Burlington, MA 01803, USA
Phone: +1 781 993 3685
Email: ram.gopal@nokia.com

Alan DeKok
IDT Inc.
1575 Carling Ave.
Ottawa, ON K1G 0T3, Canada
Phone: +1 613 724 6004 ext. 231
Email: alan.dekok@idt.com

Zsolt Haraszti
Ericsson
920 Main Campus Dr, St. 500
Raleigh, NC 27606, USA
Phone: +1 919 472 9949
Email: zsolt.haraszti@ericsson.com

Steven Blake
Ericsson
920 Main Campus Dr, St. 500
Raleigh, NC 27606, USA
Phone: +1 919 472 9913
Email: steven.blake@ericsson.com

15. Intellectual Property Right

The authors are not aware of any intellectual property right issues pertaining to this document.

16. IANA consideration

A namespace is needed to uniquely identify the LFB type in the LFB class library.

Frame type supported on input and output of LFB must also be uniquely identified.

A set of metadata supported by the LFB model must also be uniquely identified with names.