

Internet Engineering Task Force
Internet-Draft
Intended status: Experimental
Expires: January 5, 2015

Z. Cao
Q. Fu
L. Deng
China Mobile
July 4, 2014

Data Plane Processing Acceleration Framework
draft-cao-dataplane-acceleration-framework-01

Abstract

It is getting popular to running data applications over general purpose hardware/chipsets, instead of customized and dedicated hardware/chipset. This way further decouples the software functions from the hardware. But moving data processing intensive applications to general purpose hardware is still challenging, although the industry has supplied some proprietary solutions. This document discusses the problems of data plane acceleration and proposes its framework.

Status of This Memo

This Internet-Draft is submitted in full conformance with the provisions of BCP 78 and BCP 79.

Internet-Drafts are working documents of the Internet Engineering Task Force (IETF). Note that other groups may also distribute working documents as Internet-Drafts. The list of current Internet-Drafts is at <http://datatracker.ietf.org/drafts/current/>.

Internet-Drafts are draft documents valid for a maximum of six months and may be updated, replaced, or obsoleted by other documents at any time. It is inappropriate to use Internet-Drafts as reference material or to cite them other than as "work in progress."

This Internet-Draft will expire on January 5, 2015.

Copyright Notice

Copyright (c) 2014 IETF Trust and the persons identified as the document authors. All rights reserved.

This document is subject to BCP 78 and the IETF Trust's Legal Provisions Relating to IETF Documents (<http://trustee.ietf.org/license-info>) in effect on the date of publication of this document. Please review these documents carefully, as they describe your rights and restrictions with respect

to this document. Code Components extracted from this document must include Simplified BSD License text as described in Section 4.e of the Trust Legal Provisions and are provided without warranty as described in the Simplified BSD License.

Table of Contents

1. Introduction	2
2. Requirements Language	3
3. DPA Framework	3
3.1. Framework	3
3.2. Components	4
3.3. Protocol Portfolio	5
4. Existing Work - Intel DPDK	5
5. Fast Path across (Virtual) Network Functions	7
5.1. ForCES	8
6. Open Questions to IETF	8
7. Acknowledgement	9
8. IANA Considerations	9
9. Security Considerations	9
10. Informative References	9
Authors' Addresses	10

1. Introduction

The need of running network data processing functions over general purpose hardware/chipset (e.g., X86, PPC, etc) is multi-folded.

1. Decoupling software functions from hardware. Traditional network devices are built upon dedicated or deep customized hardware and chipsets. This way restricts the flexibility of both service providers and network operators.
2. Network Function Virtualization (NFV). NFV is an initiative of ETSI to virtualize the network functions to the overlay on top of the virtualization layer. It provides network elasticity in that the network functions can be scaled up/down according to the traffic load. NFV solutions often bundle with the virtual switches to provide VM-VM communications. These virtual switches are running on top of the servers that bear the network functions. Therefore, the need to accelerate the data processing efficiency is indispensable.
3. Service Time-to-Market . Via the software and hardware decoupling, the speed to provide new services (TTM) is greatly enhanced. Since more and more services would like to have the most convenient time to market, they would also like to move data processing functions on top of general purpose hardware/chipsets.

4. Capex and Opex pressure. Having the network functions running over general purpose device will help operators to cut down their Capex and Opex.
5. Cost-performance targets: software development, debug and integration is simplified; processor resource utilization is improved because the control plane and data plane can be distributed among cores with greater flexibility; development schedule risk is minimized and software maintenance is much easier with a common code base and a single development environment.

2. Requirements Language

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 [RFC2119].

3. DPA Framework

NF (Network Function): A functional building block within an operator's network infrastructure, which has well-defined external interfaces and a well-defined functional behaviour. Note that the totality of all network functions constitutes the entire network and services infrastructure of an operator/service provider. In practical terms, a Network Function is today often a network node or physical appliance. [Quoted from ETSI NFV]

3.1. Framework

The framework is depicted in Figure 1. Framework.

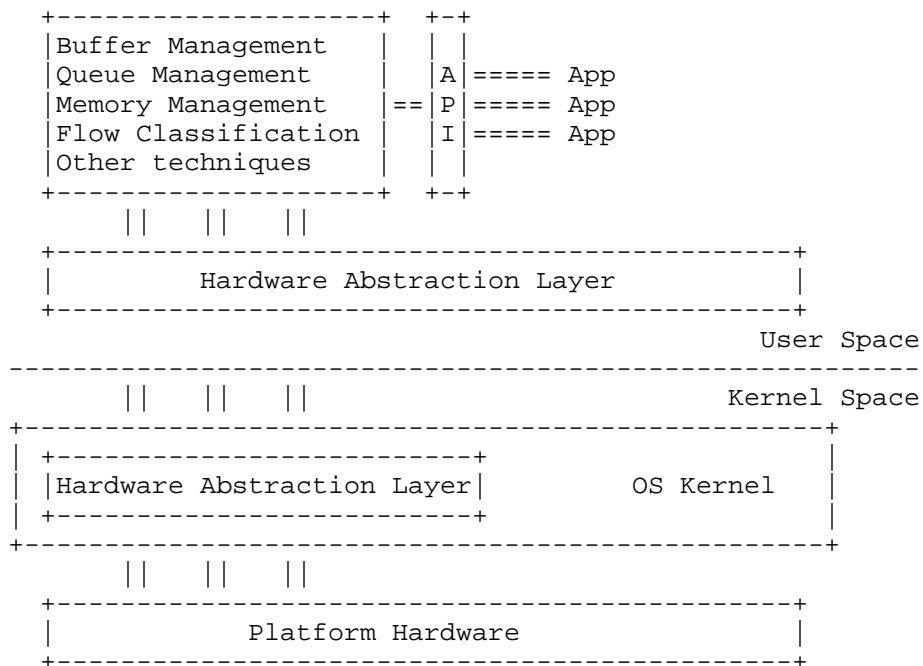


Figure 1

3.2. Components

The DPA may include the following components.

Memory/Buffer Manager. The Memory/Buffer Manager is responsible for allocating NUMA-aware pools of objects in memory and balancing memory bandwidth utilization across the channels. Such management can significantly reduce the amount of time the operating system must spend allocating and de-allocating buffers.

Queue Manager. The Queue Manager is responsible for queue scheduling. The ultimate goal of the Queue Manager is to allow different software components to process packets, while avoiding unnecessary wait times.

Flow Classification. The Flow Classification component is an efficient mechanism for generating a hash used to quickly combine packets into flows, which enables faster processing and greater throughput.

Poll Mode Drivers. The Poll Mode Drivers is capable of speeding up the packet pipeline for 1 GbE and 10 GbE ethernet controllers by

receiving and transmitting packets without the use of asynchronous, interrupt- based signaling mechanisms, which have a lot of overhead.

Environment Abstraction Layer. The Environment Abstraction Layer provides an abstraction to platform-specific initialization code, which eases application porting effort. The EAL provides access to low-level resources (hardware, memory space, logical cores, etc.) through a generic interface that hides the environment specifics from the applications and libraries.

3.3. Protocol Portfolio

On one hand, for the data plane, DPA should provide an efficient stack for common protocols utilized by various internet applications, including but not limited to:

1. Link layer: Layer 2 switch, VLAN.
2. Network layer: IPv4 and IPv6 for packet routing; MPLS and GRE/GTP for tunneled routing; IPsec, TLS/DTLS, NAT and QoS support for security and management features.
3. Transport layer: SCTP/MPTCP as well as TCP and UDP, for multi-homing/stream traffic.
4. Application layer: SSL termination for remote administration of virtualized device.

On the other hand, for the control plane, DPA should provide an efficient stack for common protocols utilized by various network devices/ISPs for improved operation and Management, including: NetFlow, sFlow, IPFIX, SPAN, RSPAN for VM traffic monitory, LACP, STP and openflow for L2/L3 management.

4. Existing Work - Intel DPDK

This section introduces DPDK [DPDK].

Intel Data Plane Development Kit (DPDK) is a set of libraries and drivers for fast packet processing on x86 platforms. It runs mostly in Linux userland. The idea of DPDK has significantly advanced the concept of consolidation of data and control planes on a general purpose processor. Such idea greatly boosts packet processing performance and throughput by providing Intel architecture-optimized libraries to accelerate L3 forwarding, yielding performance that scales linearly with the number of cores, in contrast to native Linux.

The Intel DPDK contains a growing number of libraries, whose source code is available for developers to use and/or modify in a production network element. Likewise, there are various usecase examples, such as L3 forwarding, load balancing, and timers, that help reduce development time. The libraries can be used to build applications based on "run-to completion" or "pipeline" models, enabling the equipment provider's application to maintain complete control.

the Intel DPDK software is also available to aid in the development of I/O intensive applications running in a virtualized environment. This combination allows application developers to achieve near-native performance.

The Intel DPDK provides a simple framework for fast packet processing in data plane application. Developers may use the code to understand some of the techniques employed, to build upon for prototyping, or to add their own protocol stacks. SR-IOV features are also used for hardware-based I/O sharing in I/O virtualization (IOV) mode. Therefore, it is possible to partition intel 82599 10 Gb Ethernet controller NIC resources logically and expose them to a VM as a virtual function

Furthermore, 6WIND has developed a number of value-added enhancements to the Intel DPDK library that provide increased system functionality and performance compared to the baseline software. These value-added enhancements include the following aspects.

High-performance software crypto support, implemented via the Intel Advanced Encryption Standard New Instructions (Intel AES-NI) in the Intel Xeon processor E5600 series and E5-2600 v2 series.

Device monitoring and statistics functions, such as Linux Ethtool MTU support, full RX/TX queue statistics and CRC error statistics, which enable improved system-level profiling, analysis and debug.

Support for additional Network Interface Cards (NICs), such as the Intel 82571EB Gigabit Ethernet controller, beyond those supported in the baseline Intel DPDK library.

6WIND also provides a range of optional add-on extensions to the Intel DPDK designed to improve the cost/performance of both physical and virtual networking appliances while enabling the use of the intel DPDK in software-defined networks. These optional add-ons include:

IPsec acceleration, achieved through integration of the Intel Multi-buffer Crypto for IPsec library;

Crypto acceleration via support of an external accelerator, the Intel Communications Chipset 89xx series, which is part of Intel's next-generation communications platform, codenamed "Crystal Forest"

Virtualization-related enhancements that maximize system performance by removing key I/O and communication bottlenecks include:

1. I/O Virtualization(IOV), an industry-standard approach for increasing the performance of virtual network appliances by bypassing the virtual switch within the hypervisor, thus removing the I/O performance constraints imposed by the virtual switch.
2. A virtual NIC(vNIC) driver that leverages communication between virtual machines via the virtual switch, enabling the efficient development and provisioning of systems with multiple VMs and significant East-West network traffic.
3. For system that require the ultimate level of performance for East- West traffic between VMs, a VM-to-VM driver enables direct VM-to-VM communication, bypassing the virtual switch while remaining fully compatible with industry-standard hypervisors.

These Intel DPDK enhancements and optional add-ons are maintained by 6WIND as private branch, regularly synchronized with Intel's on-going releases of the baseline library. They are delivered to customers either as a stand alone library or, for applications that also require high- performance packet processing software, and integrated within the 6WINDGate software solution.

The 6WINDGate packet processing software is designed to solve the problem of exploiting the potential packet processing performance of multicore processor through a fast pth-based architecture, while incorporating a comprehensive set of high performance networking protocols fully optimized for intel Xeon processor-based platforms.

5. Fast Path across (Virtual) Network Functions

Previous sections basically talk about the data path acceleration on one device with multiple threads/VMs sharing the physical resource. This section will talk about the data plane acceleration across multiple (virtual) network functions.

In NFV, layer 4-7 network functions are virtualized on top of the computing nodes. But sometimes, these vNFs are only used for session establishment, after which the packets can be handled by the L2/3 devices. Given that the higher layer the packet is being processed, the more challenge to its performance. So in some scenarios, it is desirable to offload the packet processing to the L2/3 fabrics,

eliminating the burden on the higher layer NFs. The scenario is depicted in Figure 2.

One vivid example is the ACL or Parental Control services. The ACL Network Function will determine the forwarding rules configured by its user, say, IP 5 tuples. After the session has been established, the ACL NF can inform the L2/3 devices about the forwarding rule in a control message. And the followed packets will be handled according to the logics.

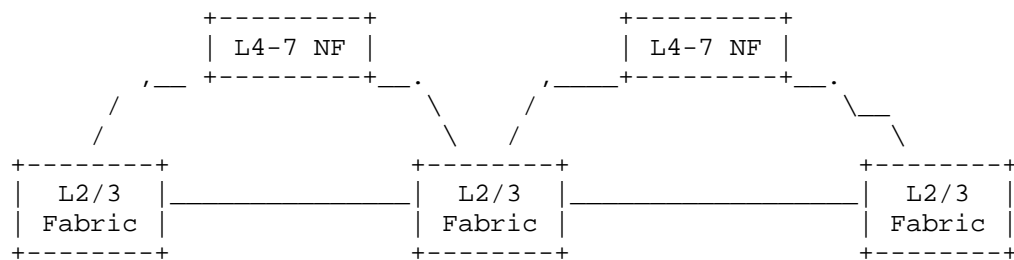


Figure 2: Fast Path across devices

5.1. ForCES

Forwarding and Control Element Separation (ForCES) [RFC5810][I-D.ietf-forces-protoextension] defines an architectural framework and associated protocols to standardize information exchange between the control plane and the forwarding plane .

In the Fast path offload senario described above, the ForCES protocols could be used or extended to serve as the communicaton protocols between the NF and L2/3 fabrics.

6. Open Questions to IETF

IETF has been design Layer 2&3 protocols, and most of them are dedicated to data plane processing. The efficient implementation of protocol and tailoring them for specific hardware/chipsets have not been considered as main-stream IETF work (there are indeed some thread anyway, e.g. tailor for M2M). But to make IETF protocols as efficient as possible is definitely within the scope of IETF. Below are some discussion of open questions to IETF w.r.t. the data plane process acceleration topic.

1. Importance. The game changing initiatives already started. NFV and further virtualization and decoupling practices are happening. Before the questions have been ported to specialized

hardware, but now the industry is changing the game. Do it need the standardization collaboration?

2. Relevance. As we authors believe it, to make IETF protocols as efficient as possible is definitely within the scope of IETF. Although implementation techniques are mostly software engineering practice and have no business with any SDOs, the abstract API design and exposure of lower layer capability will definitely benefit the data plane processing efficiency.
3. Necessity. Now that DPDK is already open source. But the experience in DPDK can feedback to IETF on how to improve the protocol design in promoting data plane acceleration effectiveness.

7. Acknowledgement

This work was inspired by the DPDK open source project.

Thank you for the discussion with Hui Deng, Dapeng Liu, and Lingli Deng on how to improve and promote this document.

8. IANA Considerations

To be specified.

9. Security Considerations

TBD.

10. Informative References

- [DPDK] "Packet Processing - Intel DPDK, <https://01.org/packet-processing/overview/dpdk-detail>", .
- [I-D.ietf-forces-protoextension] Salim, J., "ForCES Protocol Extensions", draft-ietf-forces-protoextension-02 (work in progress), June 2014.
- [NFVE2E] "Network Functions Virtualisation: End to End Architecture, <http://docbox.etsi.org/ISG/NFV/70-DRAFT/0010/NFV-0010v016.zip>", .
- [RFC2119] Bradner, S., "Key words for use in RFCs to Indicate Requirement Levels", BCP 14, RFC 2119, March 1997.

[RFC5810] Doria, A., Hadi Salim, J., Haas, R., Khosravi, H., Wang, W., Dong, L., Gopal, R., and J. Halpern, "Forwarding and Control Element Separation (ForCES) Protocol Specification", RFC 5810, March 2010.

Authors' Addresses

Zhen Cao
China Mobile
Xuanwumenxi Ave. No. 32
Beijing 100053
China

Email: zehn.cao@gmail.com, caozhen@chinamobile.com

Qiao Fu
China Mobile
Xuanwumenxi Ave. No. 32
Beijing 100053
China

Email: fuqiao@chinamobile.com

Lingli Deng
China Mobile
Xuanwumenxi Ave. No. 32
Beijing 100053
China

Email: denglingli@chinamobile.com

Internet Engineering Task Force
Internet-Draft
Intended status: Experimental
Expires: April 13, 2015

E. Haleplidis
University of Patras
J. Halpern
Ericsson
October 10, 2014

ForCES Packet Parallelization
draft-ietf-forces-packet-parallelization-03

Abstract

Forwarding and Control Element Separation (ForCES) defines an architectural framework and associated protocols to standardize information exchange between the control plane and the forwarding plane in a ForCES Network Element (ForCES NE). RFC5812 has defined the ForCES Model provides a formal way to represent the capabilities, state, and configuration of forwarding elements within the context of the ForCES protocol (RFC 5810), so that Control Elements (CEs) can control the Forwarding Elements (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.

Many network devices support parallel packet processing. This document describes how ForCES can model a network device's parallelization datapath.

Status of This Memo

This Internet-Draft is submitted in full conformance with the provisions of BCP 78 and BCP 79.

Internet-Drafts are working documents of the Internet Engineering Task Force (IETF). Note that other groups may also distribute working documents as Internet-Drafts. The list of current Internet-Drafts is at <http://datatracker.ietf.org/drafts/current/>.

Internet-Drafts are draft documents valid for a maximum of six months and may be updated, replaced, or obsoleted by other documents at any time. It is inappropriate to use Internet-Drafts as reference material or to cite them other than as "work in progress."

This Internet-Draft will expire on April 13, 2015.

Copyright Notice

Copyright (c) 2014 IETF Trust and the persons identified as the document authors. All rights reserved.

This document is subject to BCP 78 and the IETF Trust's Legal Provisions Relating to IETF Documents (<http://trustee.ietf.org/license-info>) in effect on the date of publication of this document. Please review these documents carefully, as they describe your rights and restrictions with respect to this document. Code Components extracted from this document must include Simplified BSD License text as described in Section 4.e of the Trust Legal Provisions and are provided without warranty as described in the Simplified BSD License.

Table of Contents

1. Introduction	3
1.1. Requirements Language	3
1.2. Definitions	3
2. Packet Parallelization	4
2.1. Core parallelization LFB	6
2.2. Parallelization metadata	9
3. Parallel Base Types	9
3.1. Frame Types	9
3.2. Data Types	10
3.3. MetaData Types	10
4. Parallel LFBs	11
4.1. Splitter	11
4.1.1. Data Handling	11
4.1.2. Components	12
4.1.3. Capabilities	12
4.1.4. Events	12
4.2. Merger	12
4.2.1. Data Handling	13
4.2.2. Components	14
4.2.3. Capabilities	14
4.2.4. Events	14
4.3. CoreParallelization	15
4.3.1. Data Handling	15
4.3.2. Components	15
4.3.3. Capabilities	15
4.3.4. Events	15
5. XML for Parallel LFB library	16
6. Acknowledgments	23
7. IANA Considerations	24
7.1. LFB Class Names and LFB Class Identifiers	24
7.2. Metadata ID	25

8. Security Considerations	25
9. References	25
9.1. Normative References	25
9.2. Informative References	26
Authors' Addresses	26

1. Introduction

A lot of network devices can process packets in a parallel manner. The Forwarding and Control Element Separation (ForCES) Model [RFC5812] presents a formal way to describe the Forwarding Plane's datapath with Logical Function Blocks (LFBs) using XML. This document describes how packet parallelization can be described with the ForCES model.

The modeling concept has been influenced by Cilk [Cilk]. Cilk is a programming language that has been developed since 1994 at the MIT Laboratory to allow programmers to identify elements that can be executed in parallel. The two Cilk concepts used in this document is spawn and sync. Spawn being the place where parallel tasks can start and sync being the place where the parallel task finishes and must collect all parallel output.

As task, we define a grouping of packets or pieces of a packet (chunks) that belong to the same original packet and are going to be processed in parallel. All packets/chunks of the same task will be distinguished by an identifier, in the specific case we use a 32-bit identifier named task correlator.

This document is in the experimental track and thus the LFB Class IDs will not be included in the standard action's values. Therefore the LFB Class IDs must have a value larger than 65535 and the LFB names must begin with the prefix 'Ext-'. However for brevity, when we refer to the LFB Class names in the text of this document (not the formal definitions), the 'Ext-' prefix will be omitted.

1.1. Requirements Language

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 [RFC2119].

1.2. Definitions

This document follows the terminology defined by the ForCES Model in [RFC5812]. In particular, the reader is expected to be familiar with the following terms:

FE

CE

FE Model

LFB (Logical Functional Block) Class (or type)

LFB Instance

LFB Model

Element

Attribute

LFB Metadata

ForCES Component

LFB Class Library

This document also introduces the following terms:

Chunk - Pieces of a packet

Task - Grouping of packets or chunks belong to the same packet that are processed in parallel

Task Correlator - A 32-bit identifier that uniquely distinguishes tasks

Split Type - A parallel type where the packets are split into chunks to be processed in parallel. Each task in a split type is composed only of chunks.

Flood Type - A parallel type where the packets are copied as is to downstream LFBs to be processed in parallel. Each task in a flood type is composed only of packets.

2. Packet Parallelization

This document addresses the following two types of packet parallelization:

1. Flood - where a copy of a packet is sent to multiple LFBs to be processed in parallel.

2. Split - where the packet will be split in equal size chunks specified by the CE and sent to multiple LFB instances probably of the same LFB class to be processed in parallel.

It must be noted that the process of copying the packet in the Flood parallel type is implementation dependent and is loosely defined here. An implementer may either decide to physical copy the packet and send all packets on the parallel paths, or may decide to logically copy the packet by simply sending, for example, pointers of the same packet provided that the necessary interlocks are taken into account. The implementer has to take into account the device's characteristics to decide which approach fits best to the device.

In the split parallel type, while harder, the implementer may also decide to logically split the packet and send, for example, pointers to parts of the packet, provided that the necessary interlocks are managed. In addition, how chunks are distributed to the LFBs, e.g., which chunk to which LFB, is implementation dependent. For example while usually chunks are sent to the same LFB class, the number of LFB instances may not equal to the number of chunks. It is up to the implementer to decide how these chunks will be sent, for example in a round-robin fashion.

This document introduces two LFBs that are used in before and after the parallelization occurs:

1. Splitter - similar to Cilk's spawn. An LFB that will split the path of a packet which will be sent to multiple downstream LFBs to be processed in parallel.
2. Merger - similar to Cilk's sync. An LFB that will receive packets or chunks of the same initial packet and merge them and the results into one packet.

Both parallel packet distribution types can currently be achieved with the ForCES model. The splitter LFB has one group output that produces either chunks or packets to be sent to LFBs for processing and the merger LFB has one group input that expects either packets or chunks to aggregate all the parallel packets or chunks and produce a single packet. Figure 1 shows a simple example of a split parallel datapath along with the splitter and merger LFB. Figure 2 shows an example of a flood parallel datapath along with the splitter and merger LFB.

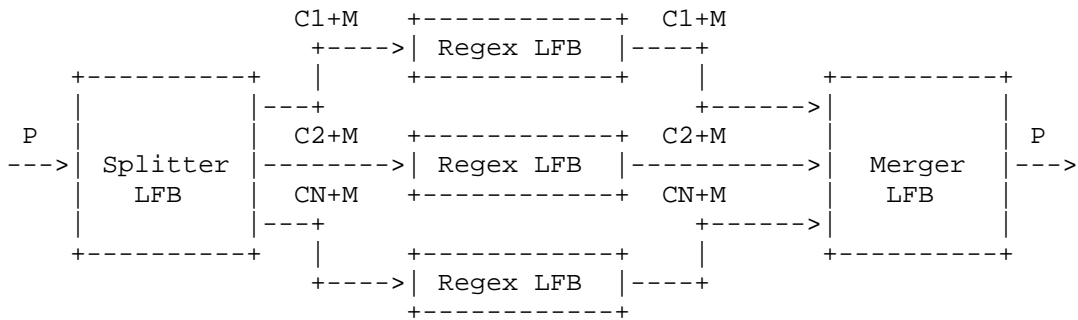


Figure 1: Simple split parallel processing

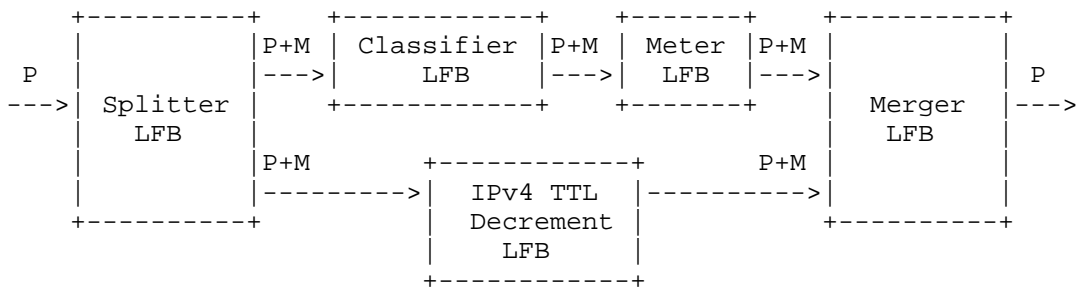


Figure 2: Simple flood parallel processing

This version of the modeling framework does not allow for nested parallel datapath topologies. This decision was reached by the authors and the ForCES working group as there was no strong use case or need at decision time. This led to a simpler metadata definition, which was needed to be transported between the splitter and the corresponding merger. If there is a need for nested parallel datapaths a new version of a splitter and merger will be needed to be defined as well as an augmentation to the defined metadata.

2.1. Core parallelization LFB

One important element to a developer is the ability to define which LFBs can be used in a parallel mode, which LFBs can be parallelized with which as well as the order in which parallel LFBs can be assembled.

To access the parallelization details, we opted for defining a new LFB class - the CoreParallelization LFB. This choice was an alternative to making another change to the core FEObject LFB. The CoreParallelization exists merely to define the capabilities for an FE's LFB parallelization. A CE using the ForCES Protocol [RFC5810]

can check the existence of this LFB class in the FEObject's SupportedLFBs component. The existence of the CoreParallelization LFB will indicate to the CE that the specific FE supports parallelization. There MUST be only one instance of the CoreParallelization LFB per FE.

The topology of the parallel datapath can be deferred and manipulated from the FEObject LFB's LFBTopology.

The CoreParallelization requires only one capability in order to specify each LFB that can be used in a parallel mode:

- o The Name of the LFB.
- o The Class ID of the LFB.
- o The Version of the LFB.
- o The number of instances that class can support in parallel.
- o A list of LFB classes that can follow this LFB class in a pipeline for a parallel path.
- o A list of LFB classes that can exist before this LFB class in a pipeline for a parallel path.
- o A list of LFB classes that can process packets or chunks in parallel with this LFB class.

```
<!-- Datatype -->
<dataTypeDef>
  <name>ParallelLFBType</name>
  <synopsis>Table entry for parallel LFBs</synopsis>
  <struct>
    <component componentID="1">
      <name>LFBName</name>
      <synopsis>The name of an LFB Class</synopsis>
      <typeRef>string</typeRef>
    </component>
    <component componentID="2">
      <name>LFBClassID</name>
      <synopsis>The id of the LFB Class</synopsis>
      <typeRef>uint32</typeRef>
    </component>
    <component componentID="3">
      <name>LFBVersion</name>
      <synopsis>The version of the LFB Class used by this FE
    </synopsis>
```

```

        <typeRef>string</typeRef>
    </component>
    <component componentID="4">
        <name>LFBParallelOccurrenceLimit</name>
        <synopsis>The upper limit of instances of the same
            parallel LFBs of this class</synopsis>
        <optional />
        <typeRef>uint32</typeRef>
    </component>
    <component componentID="5">
        <name>AllowedParallelAfters</name>
        <synopsis>List of LFB Classes that can follow this LFB
            in a parallel pipeline</synopsis>
        <optional />
        <array>
            <typeRef>uint32</typeRef>
        </array>
    </component>
    <component componentID="6">
        <name>AllowedParallelBefores</name>
        <synopsis>List of LFB Classes that this LFB class can
            follow in a parallel pipeline</synopsis>
        <optional />
        <array>
            <typeRef>uint32</typeRef>
        </array>
    </component>
    <component componentID="7">
        <name>AllowedParallel</name>
        <synopsis>List of LFB Classes that this LFB class be run
            in parallel with</synopsis>
        <array>
            <typeRef>uint32</typeRef>
        </array>
    </component>
</struct>
</dataTypeDef>

<!-- Capability -->
    <capability componentID="32">
        <name>ParallelLFBs</name>
        <synopsis>List of all supported parallel LFBs</synopsis>
        <array type="Variable-size">
            <typeRef>ParallelLFBType</typeRef>
        </array>
    </capability>

```

Figure 3: XML Definitions for CoreParallelization LFB

2.2. Parallelization metadata

It is expected that the splitting and merging mechanisms are an implementation issue. This document plays the role of defining the operational parameters for the splitting and merging, namely, the size of the chunks, what happens if a packet or chunk has been marked as invalid and whether the merge LFB should wait for all packets or chunks to arrive. The following metadata set is defined as a struct:

1. ParallelType - Flood or split
2. TaskCorrelator - Identify packets or chunks that belonged to the initial packet that entered the Splitter LFB
3. ParallelNum - Sequence Number of the packet or the chunk for a specific task.
4. ParallelPartsCount - Total number of packets or chunks for a specific task.

This metadata is produced from the Splitter LFB and is opaque to LFBs in parallel paths and is passed along to the merger LFB without being consumed.

In the case in which an LFB decides that a packet/chunk has to be dropped, the LFB MAY drop the packet/chunk but the metadata MUST be sent to the Merger LFB's InvalidIn input port for merging purposes.

Additional metadata produced by LFBs inside a datapath MAY be aggregated within the Merger LFB and sent on after the merging process. In case of receiving the same metadata definition with multiple values the merger LFB MUST keep the first received from a valid packet or chunk.

3. Parallel Base Types

3.1. Frame Types

One frame type has been defined in this library.

Frame Type Name	Synopsis
Chunk	A chunk is a frame that is part of an original larger frame

Parallel Frame Types

3.2. Data Types

One data type has been defined in this library.

Data Type Name	Type	Synopsis
ParallelTypes	Atomic uchar. Special Values Flood (0), Split (1).	The type of parallelization this packet will go through

Parallel Data Types

3.3. MetaData Types

The following metadata structure with ID 16, using the ForCES model extension [I-D.ietf-forces-model-extension], is defined for the parallelization library:

Metadata Name	Type	ID	Synopsis
ParallelType	uchar	1	The type of parallelization this packet will go through. 0 for flood, 1 for split.
TaskCorrelator	uint32	2	An identification number to specify that a packet or a chunks belongs to the same parallel task.
ParallelNum	uint32	3	Defines the number of a specific packet or chunk of a specific task.
ParallelPartsCount	uint32	4	Defines the total number of packets or chunks for a specific task.

Metadata Structure for Merging

4. Parallel LFBs

4.1. Splitter

The splitter LFB takes part in parallelizing the processing datapath by sending either the same packet Figure 2 or chunks Figure 1 of the same packet to multiple LFBs.

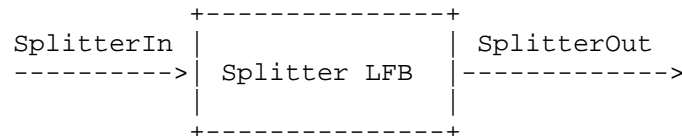


Figure 4: Splitter LFB

4.1.1. Data Handling

The splitter LFB receives any kind of packet via the singleton input, Input. Depending upon the CE's configuration of the ParallelType component, if the parallel type is of type flood (0), the same packet MUST be sent through all of the group output SplitterOut's instances. If the parallel type is of type split (1), the packet will be split into same size chunks except the last which MAY be smaller, with the max size being defined by the ChunkSize component. Chunks MAY be

sent out in a round-robin fashion through the group output ParallelOut's instances or in any other way defined by the implementer. Each packet or chunk will be accompanied by the following metadata set as a struct:

- o ParallelType - The parallel type, split or flood.
- o ParallelID - generated by the splitter LFB to identify that chunks or packets belong to the same parallel task.
- o ParallelNum - each chunk or packet of a parallel id will be assigned a number in order for the merger LFB to know when it has gathered them all along with the ParallelPartsCount metadata.
- o ParallelPartsCount - the number of chunks or packets for the specific task.

4.1.2. Components

The splitter LFB has only two components. The first is the ParallelType, an uint32 that defines how the packet will be processed by the Splitter LFB. The second is the ChunkSize, an uint32 that specifies the size of each chunk when a packet is split into multiple same size chunks. The last chunk MAY be smaller than the value of the ChunkSize.

4.1.3. Capabilities

This LFB has only one capability specified, the MinMaxChunkSize a struct of two uint32 to specify the minimum and maximum chunk size.

4.1.4. Events

This LFB has no events specified.

4.2. Merger

The merger LFB is the synchronization point for multiple packets or packet chunks of the same task, emanating out of the parallel path as illustrated in Figure 2 and Figure 1.

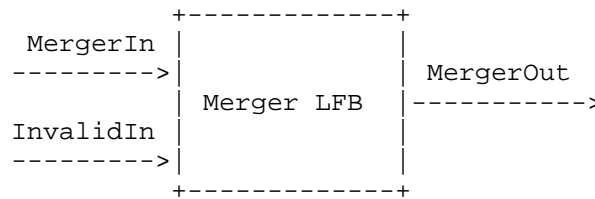


Figure 5: Merger LFB

4.2.1. Data Handling

The merger LFB receives either a packet or a chunk via the group input ParallelIn, along with the ParallelType metadata that, the TaskCorrelator, the ParallelNum and the ParallelPartsCount.

In case that an upstream LFB has dropped a packet or a chunk the merger LFB MAY receive only the metadata or both metadata and packet or chunk through the InvalidIn group input port. It SHOULD receive a metadata specifying the error code. Currently defined metadata's in the Base LFB Library [RFC6956] are the ExceptionID and the ValidateErrorID.

If the MergeWaitType is set to false the Merger LFB will initiate the merge process upon receiving the first packet. If false, for each task identified by the task correlator, it will wait for all packets/chunks to arrive unless the MergeWaitTimeoutTimer timer expires. If the MergeWaitTimeoutTimer has expired, the Merger MUST consider the rest of the packets/chunks, that have not been received, as invalid and MUST handle the packets according to the InvalidAction value.

If one packet or chunk has been received through the InvalidIn port then the merging procedure will handle the packets/chunks according to the InvalidAction value. If the InvalidAction component has been set to 0 then if one packet or chunk is not valid all will dropped, else the process will initiate. Once the merging process has been completed the resulting packet will be sent via the singleton output port MergerOut.

If the Merger LFB receives different values for the same metadata from different packets or chunks that has the same task correlator then the Merger LFB will use the first metadata from a packet or chunk that entered the LFB through the MergerIn input port.

4.2.2. Components

This LFB has the following components specified:

1. InvalidAction - a uchar defining what the Merge LFB will do if an invalid chunk or packet is received. If set to 0 (DropAll) the merge will be considered invalid and all chunks or packets will be dropped. If set to 1 (Continue) the merge will continue.
2. MergeWaitTimeoutTimer - a uint32 defining the amount of time, in milliseconds, that the Merger will wait for all packets or chunks within the same task to arrive before considering them invalid. The MergeWaitTimeoutTimer starts as soon as the first chunk or packet of a parallel task arrives.
3. MergeWaitType - a boolean. If true the Merger LFB will wait for all packets or chunks to be received prior to performing the merge. If false, when one packet or a chunk with a response is received by the merge LFB it will start with the merge process.
4. InvalidMergesCounter - a uint32 that counts the number of merges where there is at least one packet or chunk that entered the merger LFB through the InvalidIn input port.
5. InvalidTotalCounter - a uint32 that counts the number of merges where all packets/chunks entered the merger LFB through the InvalidIn input port.
6. InvalidIDCounters - a struct of two arrays. Each array has a uint32 per row. Each array counts number of invalid merges where at least one packet or chunk entered through InvalidID per error ID. The first array is the InvalidExceptionID and the second is the InvalidValidateErrorID.

4.2.3. Capabilities

This LFB has no capabilities specified.

4.2.4. Events

This LFB specifies only two event. The first detects whether the InvalidMergesCounter has exceeded a specific value and the second detects whether the InvalidAllCounter has exceeded a specific value. Both error reports will send the respective counter value. Event Filters can be used to limit the number of messages

4.3. CoreParallelization

A core LFB that specifies that the FE supports parallelization, instead of updating the FEObject LFB

4.3.1. Data Handling

The CoreParallelization does not handle data.

4.3.2. Components

This LFB has no components specified.

4.3.3. Capabilities

This LFB has only one capability specified. The ParallelLFBs is a table which lists all the LFBs that can be parallelized. Each row of the table contains:

1. LFBName - a string. The Name of the parallel LFB.
2. LFBClassID - a uint32. The Class ID of the parallel LFB.
3. LFBVersion - a string. The Version of the parallel LFB.
4. LFBParallelOccurrenceLimit - a uint32. The upper limit of instances of the same parallel LFBs of this class.
5. AllowedParallelAfters - a table of uint32s (LFB Class IDs). A list of LFB classes that can follow this LFB class in a pipeline for a parallel path.
6. AllowedParallelBefores - a table of uint32s (LFB Class IDs). A list of LFB classes that can exist before this LFB class in a pipeline for a parallel path.
7. AllowedParallel - a table of uint32s (LFB Class IDs). A list of LFB classes that can process packets or chunks in parallel with this LFB class.

4.3.4. Events

This LFB specifies no events

5. XML for Parallel LFB library

```
<?xml version="1.0" encoding="UTF-8"?>
<LFBLibrary xmlns="urn:ietf:params:xml:ns:forces:lfbmodel:1.1"
  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  xsi:schemaLocation="urn:ietf:params:xml:ns:forces:lfbmodel:1.1"
  provides="Parallel">
  <load library="BaseTypeLibrary" location="BaseTypeLibrary.LFB"/>
  <frameDefs>
    <frameDef>
      <name>Chunk</name>
      <synopsis>A chunk is a frame that is part of an original
        larger frame</synopsis>
    </frameDef>
  </frameDefs>
  <dataTypeDefs>
    <dataTypeDef>
      <name>ParallelTypes</name>
      <synopsis>The type of parallelization this packet will go
        through</synopsis>
      <atomic>
        <baseType>uchar</baseType>
        <specialValues>
          <specialValue value="0">
            <name>Flood</name>
            <synopsis>The packet/chunk has been sent as a whole
              to multiple recipients</synopsis>
          </specialValue>
          <specialValue value="1">
            <name>Split</name>
            <synopsis>The packet/chunk has been split into
              multiple chunks and sent to recipients</synopsis>
          </specialValue>
        </specialValues>
      </atomic>
    </dataTypeDef>
    <dataTypeDef>
      <name>ParallelLFBType</name>
      <synopsis>Table entry for parallel LFBs</synopsis>
      <struct>
        <component componentID="1">
          <name>LFBName</name>
          <synopsis>The name of an LFB Class</synopsis>
          <typeRef>string</typeRef>
        </component>
        <component componentID="2">
          <name>LFBClassID</name>
          <synopsis>The id of the LFB Class</synopsis>
```

```

        <typeRef>uint32</typeRef>
    </component>
    <component componentID="3">
        <name>LFBVersion</name>
        <synopsis>The version of the LFB Class used by this FE
        </synopsis>
        <typeRef>string</typeRef>
    </component>
    <component componentID="4">
        <name>LFBParallelOccurrenceLimit</name>
        <synopsis>The upper limit of instances of the same
        parallel LFBs of this class</synopsis>
        <optional/>
        <typeRef>uint32</typeRef>
    </component>
    <component componentID="5">
        <name>AllowedParallelAfters</name>
        <synopsis>List of LFB Classes that can follow this LFB
        in a parallel pipeline</synopsis>
        <optional/>
        <array>
            <typeRef>uint32</typeRef>
        </array>
    </component>
    <component componentID="6">
        <name>AllowedParallelBefores</name>
        <synopsis>List of LFB Classes that this LFB class can
        follow in a parallel pipeline</synopsis>
        <optional/>
        <array>
            <typeRef>uint32</typeRef>
        </array>
    </component>
    <component componentID="7">
        <name>AllowedParallel</name>
        <synopsis>List of LFB Classes that this LFB class be run
        in parallel with</synopsis>
        <array>
            <typeRef>uint32</typeRef>
        </array>
    </component>
</struct>
</dataTypeDef>
</dataTypeDefs>
<metadataDefs>
    <metadataDef>
        <name>ParallelMetadataSet</name>
        <synopsis>A metadata Set for parallelization related LFBs

```

```

    </synopsis>
<metadataID>32</metadataID>
<struct>
  <component componentID="1">
    <name>ParallelType</name>
    <synopsis>The type of parallelization this packet/chunk
      has gone through</synopsis>
    <typeRef>ParallelTypes</typeRef>
  </component>
  <component componentID="2">
    <name>TaskCorrelator</name>
    <synopsis>An identification number to specify that
      packets or chunks originate from the same packet.
    </synopsis>
    <typeRef>uint32</typeRef>
  </component>
  <component componentID="3">
    <name>ParallelNum</name>
    <synopsis>Defines the number of the specific packet or
      chunk of the specific parallel ID.</synopsis>
    <typeRef>uint32</typeRef>
  </component>
  <component componentID="4">
    <name>ParallelPartsCount</name>
    <synopsis>Defines the total number of packets or chunks
      for the specific parallel ID.</synopsis>
    <typeRef>uint32</typeRef>
  </component>
</struct>
</metadataDef>
</metadataDefs>
<LFBClassDefs>
  <LFBClassDef LFBClassID="65537">
    <name>Ext-Splitter</name>
    <synopsis>A splitter LFB takes part in parallelizing the
      processing datapath. It will either send the same packet
      or chunks of one packet to multiple LFBs</synopsis>
    <version>1.0</version>
    <inputPorts>
      <inputPort>
        <name>SplitterIn</name>
        <synopsis>An input port expecting any kind of frame
        </synopsis>
        <expectation>
          <frameExpected>
            <ref>Arbitrary</ref>
          </frameExpected>
        </expectation>
      </inputPort>
    </inputPorts>
  </LFBClassDef>
</LFBClassDefs>

```

```
</inputPort>
</inputPorts>
<outputPorts>
  <outputPort group="true">
    <name>SplitterOut</name>
    <synopsis>A parallel output port that sends the same
              packet to all output instances or chunks of the same
              packet different chunk on each instance.</synopsis>
    <product>
      <frameProduced>
        <ref>Arbitrary</ref>
        <ref>Chunk</ref>
      </frameProduced>
      <metadataProduced>
        <ref>ParallelMetadataSet</ref>
      </metadataProduced>
    </product>
  </outputPort>
</outputPorts>
<components>
  <component componentID="1" access="read-write">
    <name>ParallelType</name>
    <synopsis>The type of parallelization this packet will
              go through</synopsis>
    <typeRef>ParallelTypes</typeRef>
  </component>
  <component componentID="2" access="read-write">
    <name>ChunkSize</name>
    <synopsis>The size of a chunk when a packet is split
              into multiple same size chunks</synopsis>
    <typeRef>uint32</typeRef>
  </component>
</components>
<capabilities>
  <capability componentID="31">
    <name>MinMaxChunkSize</name>
    <synopsis>The minimum and maximum size of a chunk
              capable of split by this LFB</synopsis>
    <struct>
      <component componentID="1">
        <name>MinChunkSize</name>
        <synopsis>Minimum chunk size</synopsis>
        <optional/>
        <typeRef>uint32</typeRef>
      </component>
      <component componentID="2">
        <name>MaxChunkSize</name>
        <synopsis>Maximum chunk size</synopsis>
```

```
        <typeRef>uint32</typeRef>
      </component>
    </struct>
  </capability>
</capabilities>
</LFBClassDef>
<LFBClassDef LFBClassID="65538">
  <name>Ext-Merger</name>
  <synopsis>A merger LFB receives multiple packets or multiple
    chunks of the same packet and merge them into one merged
    packet</synopsis>
  <version>1.0</version>
  <inputPorts>
    <inputPort group="true">
      <name>MergerIn</name>
      <synopsis>A parallel input port that accepts packets
        or chunks from all output instances</synopsis>
      <expectation>
        <frameExpected>
          <ref>Arbitrary</ref>
          <ref>Chunk</ref>
        </frameExpected>
        <metadataExpected>
          <ref>ParallelMetadataSet</ref>
        </metadataExpected>
      </expectation>
    </inputPort>
    <inputPort group="true">
      <name>InvalidIn</name>
      <synopsis>When a packet is sent out of an error port of
        an LFB in a parallel path will be sent to this
        output port in the Merger LFB</synopsis>
      <expectation>
        <frameExpected>
          <ref>Arbitrary</ref>
          <ref>Chunk</ref>
        </frameExpected>
        <metadataExpected>
          <one-of>
            <ref>ExceptionID</ref>
            <ref>ValidateErrorID</ref>
          </one-of>
        </metadataExpected>
      </expectation>
    </inputPort>
  </inputPorts>
  <outputPorts>
    <outputPort>
```

```

    <name>MergerOut</name>
    <synopsis>An output port expecting any kind of frame
    </synopsis>
    <product>
      <frameProduced>
        <ref>Arbitrary</ref>
      </frameProduced>
    </product>
  </outputPort>
</outputPorts>
<components>
  <component componentID="1" access="read-write">
    <name>InvalidAction</name>
    <synopsis>What the Merge LFB will do if an invalid
      chunk or packet is received</synopsis>
    <atomic>
      <baseType>uchar</baseType>
      <specialValues>
        <specialValue value="0">
          <name>DropAll</name>
          <synopsis>Drop all packets or chunks
            </synopsis>
        </specialValue>
        <specialValue value="1">
          <name>Continue</name>
          <synopsis>Continue with the merge</synopsis>
        </specialValue>
      </specialValues>
    </atomic>
  </component>
  <component componentID="2" access="read-write">
    <name>MergeWaitType</name>
    <synopsis>Whether the Merge LFB will wait for all
      packets or chunks to be received prior to sending
      out a response</synopsis>
    <typeRef>boolean</typeRef>
  </component>
  <component componentID="3" access="read-write">
    <name>MergeWaitTimeoutTimer</name>
    <synopsis>The time that the Merger will wait
      for all packets or chunks within the same task to arrive
      before considering them invalid.</synopsis>
    <typeRef>uint32</typeRef>
  </component>
  <component componentID="4" access="read-reset">
    <name>InvalidMergesCounter</name>
    <synopsis>Counts the number of merges where there is at
      least one packet/chunk that entered the merger LFB

```

```

        through the InvalidIn input port</synopsis>
        <typeRef>uint32</typeRef>
    </component>
    <component componentID="5" access="read-reset">
        <name>InvalidTotalCounter</name>
        <synopsis>Counts the number of merges where all
            packets/chunks entered the merger LFB through the
            InvalidIn input port</synopsis>
        <typeRef>uint32</typeRef>
    </component>
    <component componentID="6" access="read-reset">
        <name>InvalidIDCounters</name>
        <synopsis>Counts number of invalid merges where at
            least one packet/chunk entered through InvalidID per
            error ID</synopsis>
        <struct>
            <component componentID="1">
                <name>InvalidExceptionID</name>
                <synopsis>Per Exception ID</synopsis>
                <array>
                    <typeRef>uint32</typeRef>
                </array>
            </component>
            <component componentID="2">
                <name>InvalidValidateErrorID</name>
                <synopsis>Per Validate Error ID</synopsis>
                <array>
                    <typeRef>uint32</typeRef>
                </array>
            </component>
        </struct>
    </component>
</components>
<events baseID="30">
    <event eventID="1">
        <name>ManyInvalids</name>
        <synopsis>An event that specifies if there are too many
            invalids</synopsis>
        <eventTarget>
            <eventField>InvalidCounter</eventField>
        </eventTarget>
        <eventGreaterThan/>
        <eventReports>
            <eventReport>
                <eventField>InvalidMergesCounter</eventField>
            </eventReport>
        </eventReports>
    </event>

```



```

    <event eventID="2">
      <name>ManyTotalInvalids</name>
      <synopsis>An event that specifies if there are too many
        invalids</synopsis>
      <eventTarget>
        <eventField>InvalidTotalCounter</eventField>
      </eventTarget>
      <eventGreaterThan/>
      <eventReports>
        <eventReport>
          <eventField>InvalidTotalCounter</eventField>
        </eventReport>
      </eventReports>
    </event>
  </events>
</LFBClassDef>
<LFBClassDef LFBClassID="65539">
  <name>Ext-CoreParallelization</name>
  <synopsis>A core LFB that specifies that the FE supports
    parallelization, instead of updating the FEObject
    LFB</synopsis>
  <version>1.0</version>
  <capabilities>
    <capability componentID="10">
      <name>ParallelLFBs</name>
      <synopsis>A table which lists all the LFBs that can be
        parallelized</synopsis>
      <array>
        <typeRef>ParallelLFBType</typeRef>
      </array>
    </capability>
  </capabilities>
</LFBClassDef>
</LFBClassDefs>
</LFBLibrary>

```

Figure 6: Parallel LFB library

6. Acknowledgments

The authors would like to thank Edward Crabbe for the initial discussion that led to the creation of this document and Jamal Hadi Salim and Dave Hood for comments and discussions that made this document better. Additionally Adrian Farrel for his AD review. Finally Francis Dupont for his Gen-Art review and Magnus Nystroem for his security review which refined this document to its final shape.

7. IANA Considerations

7.1. LFB Class Names and LFB Class Identifiers

LFB classes defined by this document do not belong to LFBs defined by Standards Track RFCs in the <http://www.iana.org/assignments/forces-registry>. As such the values defined in this document are in the above 65535 value range.

This specification includes the following LFB class names and LFB class identifiers:

LFB Class Identifier	LFB Class Name	LFB Version	Description	Reference
65537	Ext-Splitter	1.0	A splitter LFB will either send the same packet or chunks of one packet to multiple LFBs.	This document
65538	Ext-Merger	1.0	A merger LFB receives multiple packets or multiple chunks of the same packet and merge them into one.	This document
65539	Ext-CoreParallelization	1.0	A core LFB to signify the parallelization capability	This document

Logical Functional Block (LFB) Class Names and Class Identifiers

7.2. Metadata ID

The Metadata ID namespace is 32 bits long. Values assigned by this specification are:

Value	Name	Definition
0x00000010	ParallelMetadataSet	This document

Metadata ID assigned by this specification

8. Security Considerations

This document does not alter either the ForCES model [RFC5812] or the ForCES protocol [RFC5810] and as such has no impact on their security considerations. This document simply defines the operational parameters and capabilities of LFBs that perform parallelization and not how parallelization is implemented. Finally, this document does not attempt to analyze the presence or possibility of security interactions created by allowing parallel operations on packets. Any such issues, if they exist, are for the designers of the particular data path, not the general mechanism.

9. References

9.1. Normative References

- [I-D.ietf-forces-model-extension]
Haleplidis, E., "ForCES Model Extension", draft-ietf-forces-model-extension-05 (work in progress), September 2014.
- [RFC2119] Bradner, S., "Key words for use in RFCs to Indicate Requirement Levels", BCP 14, RFC 2119, March 1997.
- [RFC5810] Doria, A., Hadi Salim, J., Haas, R., Khosravi, H., Wang, W., Dong, L., Gopal, R., and J. Halpern, "Forwarding and Control Element Separation (ForCES) Protocol Specification", RFC 5810, March 2010.
- [RFC5812] Halpern, J. and J. Hadi Salim, "Forwarding and Control Element Separation (ForCES) Forwarding Element Model", RFC 5812, March 2010.

[RFC6956] Wang, W., Haleplidis, E., Ogawa, K., Li, C., and J. Halpern, "Forwarding and Control Element Separation (ForCES) Logical Function Block (LFB) Library", RFC 6956, June 2013.

9.2. Informative References

[Cilk] MIT, "Cilk language",
<<http://supertech.csail.mit.edu/cilk/>>.

Authors' Addresses

Evangelos Haleplidis
University of Patras
Department of Electrical and Computer Engineering
Patras 26500
Greece

Email: ehalep@ece.upatras.gr

Joel Halpern
Ericsson
P.O. Box 6049
Leesburg VA 20178
USA

Phone: +1 703 371 3043
Email: joel.halpern@ericsson.com

Internet Engineering Task Force
Internet-Draft
Intended status: Informational
Expires: January 4, 2015

D. Joachimpillai
Verizon
J. Hadi Salim
Mojatatu Networks
July 3, 2014

ForCES Inter-FE LFB
draft-joachimpillai-forces-interfelfb-04

Abstract

Forwarding and Control Element Separation (ForCES) defines an architectural framework and associated protocols to standardize information exchange between the control plane and the forwarding plane in a ForCES Network Element (ForCES NE). RFC5812 has defined the ForCES Model which provides a formal way to represent the capabilities, state, and configuration of forwarding elements (FEs) within the context of the ForCES protocol. 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. The control elements (CEs) can control the FEs using the ForCES model definition.

The ForCES WG charter has been extended to allow the LFB topology to be across FEs. This document describes a non-intrusive way to extend the LFB topology across FEs.

Status of this Memo

This Internet-Draft is submitted in full conformance with the provisions of BCP 78 and BCP 79.

Internet-Drafts are working documents of the Internet Engineering Task Force (IETF). Note that other groups may also distribute working documents as Internet-Drafts. The list of current Internet-Drafts is at <http://datatracker.ietf.org/drafts/current/>.

Internet-Drafts are draft documents valid for a maximum of six months and may be updated, replaced, or obsoleted by other documents at any time. It is inappropriate to use Internet-Drafts as reference material or to cite them other than as "work in progress."

This Internet-Draft will expire on January 4, 2015.

Copyright Notice

Copyright (c) 2014 IETF Trust and the persons identified as the

document authors. All rights reserved.

This document is subject to BCP 78 and the IETF Trust's Legal Provisions Relating to IETF Documents (<http://trustee.ietf.org/license-info>) in effect on the date of publication of this document. Please review these documents carefully, as they describe your rights and restrictions with respect to this document. Code Components extracted from this document must include Simplified BSD License text as described in Section 4.e of the Trust Legal Provisions and are provided without warranty as described in the Simplified BSD License.

Table of Contents

1. Terminology and Conventions	3
1.1. Requirements Language	3
1.2. Definitions	3
2. Introduction	4
3. Problem Scope And Use Cases	5
3.1. Basic Router	5
3.1.1. Distributing The LFB Topology	7
3.2. Arbitray Network Function	8
3.2.1. Distributing The Arbitray Network Function	9
4. Proposal Overview	10
4.1. Inserting The Inter-FE LFB	10
4.2. Inter-FE connectivity	12
4.2.1. Inter-FE Ethernet connectivity	14
4.2.1.1. Inter-FE Ethernet Connectivity Issues	14
5. Detailed Description of the inter-FE LFB	15
5.1. Data Handling	16
5.1.1. Egress Processing	17
5.1.2. Ingress Processing	17
5.2. Metadata	18
5.3. Components	18
5.4. Inter-FE LFB XML Model	18
6. Acknowledgements	20
7. IANA Considerations	20
8. Security Considerations	20
9. References	20
9.1. Normative References	20
9.2. Informative References	20
Authors' Addresses	20

1. Terminology and Conventions

1.1. Requirements Language

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 [RFC2119].

1.2. Definitions

This document follows the terminology defined by the ForCES Model in [RFC5812]. The required definitions are repeated below for clarity.

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 LFB modeling of individual Logical Functional Block (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 [RFC5810].

LFB (Logical Functional Block) Class (or type) - A template that represents a fine-grained, logically separable aspect of FE processing. Most LFBs relate to packet processing in the data path. LFB classes are the basic building blocks of the FE model.

LFB Instance - As a packet flows through an FE along a data path, it flows through one or multiple LFB instances, where each LFB is an instance of a specific LFB class. Multiple instances of the same LFB class can be present in an FE's data path. Note that we often refer to LFBs without distinguishing between an 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 an LFB, plus the associated data definition. XML is used to provide a formal definition of the necessary structures for the modeling. Four types of information are defined in the LFB model. The core part of the LFB model is the LFB class definitions; the other three types of information define constructs associated with and used by the class definition. These are reusable data types, supported frame (packet) 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 the per-packet state is

implemented within actual hardware. Metadata is sent between the FE and the CE on redirect packets.

ForCES Component - A ForCES Component is a well-defined, uniquely identifiable and addressable ForCES model building block. A component has a 32-bit ID, name, type, and an optional synopsis description. These are often referred to simply as components.

LFB Component - An LFB component is a ForCES component that defines the Operational parameters of the LFBs that must be visible to the CEs.

LFB Topology - LFB topology is a representation of the logical interconnection and the placement of LFB instances along the data path within one FE. Sometimes this representation is called intra-FE topology, to be distinguished from inter-FE topology. LFB topology is outside of the LFB model, but is part of the FE model.

FE Topology - FE topology is a representation of how multiple FEs within a single network element (NE) are interconnected. Sometimes this is called inter-FE topology, to be distinguished from intra-FE topology (i.e., LFB topology). An individual FE might not have the global knowledge of the full FE topology, but the local view of its connectivity with other FEs is considered to be part of the FE model.

Service Graph - A directed graph of LFB instances whose composition delivers a packet service.

2. Introduction

In the ForCES architecture, a packet service can be modelled by composing a graph of one or more LFB instances. The reader is referred to the details in the ForCES Model [RFC5812].

The FEObject LFB capabilities in the ForCES Model [RFC5812] define component ModifiableLFBTopology which, when advertised as true by the FE, implies FE is capable of modifying the LFB graph. In such a case, the FEObject LFB table SupportedLFBs contains information about each supported LFB class that the FE supports. For each LFB class supported, additional information of how an LFB class may be connected to other LFBs is advertised. The advertised rules describe which LFB classes a specified LFB class may succeed or precede in an LFB topology. The capability of an FE can be queried by the CE upon association.

The CE may create a packet service by describing LFB instance graph connections via updating the FEObject LFBTopology component. The created topology contains information about each inter-LFB link within the FE (each link is described in an LFBLinkType dataTypeDef). The LFBLinkType component contains sufficient information to identify precisely the end points of a link of a service graph.

Often there are requirements for the packet service graph to cross FE boundaries. This could be from a desire to scale the service or need to interact with LFBs which reside in a separate FE (eg lookaside interface to a shared TCAM, an interconnected chip, or as coarse grained functionality as an external NAT FE box being part of the service graph etc).

Given that the ForCES inter-LFB architecture calls out for ability to pass metadata between LFBs, it is imperative to define mechanisms to allow passing the metadata between inter-FE LFBs (given that packet data passing is already taken care of).

The new ForCES charter allows the LFB links in a topology to be across multiple FE (inter-FE connectivity).

This document describes extending the LFB topology across FEs i.e inter-FE connectivity without needing any changes to the ForCES definitions. It focusses on using Ethernet as the interconnection as a starting point while leaving room for other protocols (such as directly on top of IP, UDP, VXLAN, etc).

3. Problem Scope And Use Cases

The scope of this document is to solve the challenge of passing ForCES defined metadata and exceptions across FEs (be they physical or virtual). To illustrate the problem scope we present two use cases where we start with a single FE running all the functionality then split it into multiple FEs.

3.1. Basic Router

A sample LFB topology Figure 1 demonstrates a service graph for delivering basic IPV4 forwarding service within one FE. Note: although the diagram shows LFB classes connecting in the graph in reality it is a graph of LFB class instances that are inter-connected.

Since the illustration is meant only as an exercise to showcase how data and metadata is sent down or upstream on a graph of LFBs, it abstracts out any ports in both directions and talks about a generic

ingress and egress LFB. Again, for illustration purposes, the diagram does not show exception or error paths. Also left out are details on Reverse Path Filtering, ECMP, multicast handling etc. In other words, this is not meant to be a complete description of an IPV4 forwarding application; for a more complete example, please refer to the LFBlib document [RFC6956] .

The output of the ingress LFB(s) coming into the IPV4 Validator LFB will have both the IPV4 packets and, depending on the implementation, a variety of ingress metadata such as offsets into the different headers, any classification metadata, physical and virtual ports encountered, tunnelling information etc. These metadata are lumped together as "ingress metadata".

Once the IPV4 validator vets the packet (example ensures that no expired TTL etc), it feeds the packet and inherited metadata into the IPV4 unicast LPM LFB.

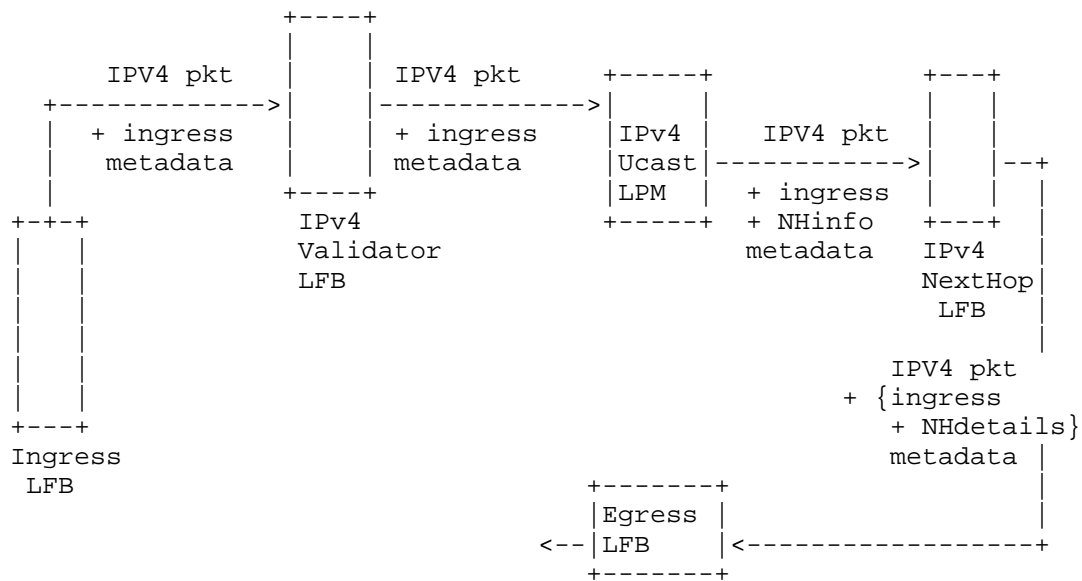


Figure 1: Basic IPV4 packet service LFB topology

The IPV4 unicast LPM LFB does a longest prefix match lookup on the IPV4 FIB using the destination IP address as a search key. The result is typically a next hop selector which is passed downstream as metadata.

The Nexthop LFB receives the IPv4 packet with an associated next hop info metadata. The NextHop LFB consumes the NH info metadata and derives from it a table index to look up the next hop table in order to find the appropriate egress information. The lookup result is used to build the next hop details to be used downstream on the egress. This information may include any source and destination information (MAC address to use, if ethernet;) as well egress ports. [Note: It is also at this LFB where typically the forwarding TTL decrement and IP checksum recalculation occurs.]

The details of the egress LFB are considered out of scope for this discussion. Suffice it is to say that somewhere within or beyond the Egress LFB the IPV4 packet will be sent out a port (ethernet, virtual or physical etc).

3.1.1. Distributing The LFB Topology

Figure 2 demonstrates one way the router LFB topology in Figure 1 may be split across two FEs (eg two ASICs). Figure 2 shows the LFB topology split across FEs after the IPV4 unicast LPM LFB.

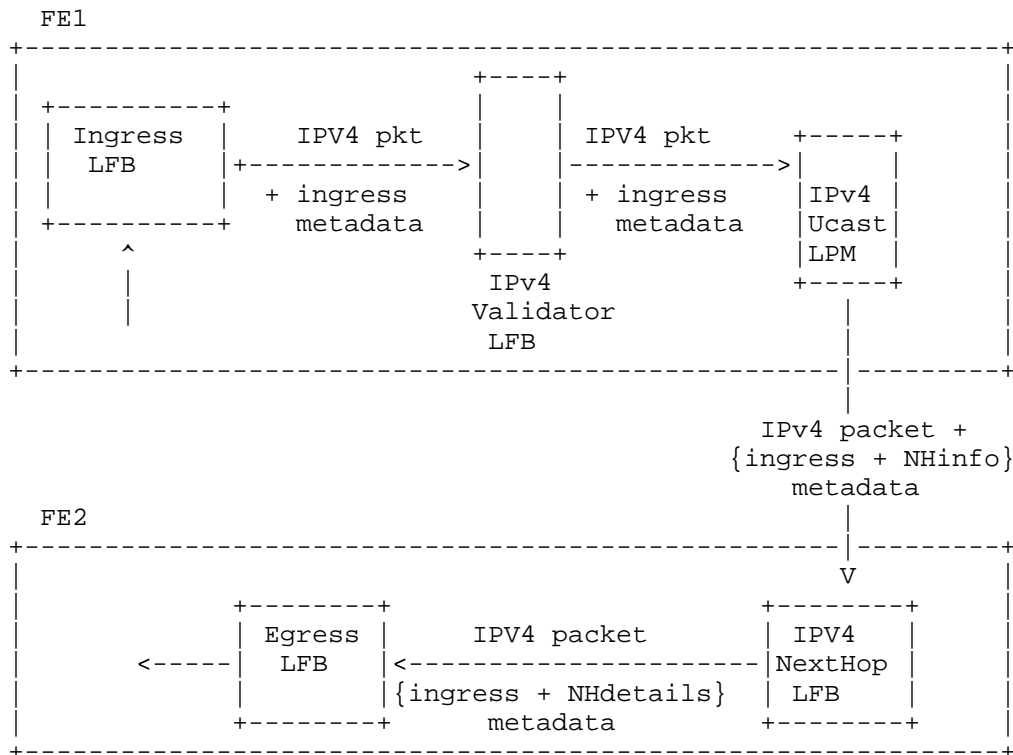


Figure 2: Split IPv4 packet service LFB topology

Some proprietary inter-connect (example Broadcom Higi over XAUI (XXX: ref needed)) maybe exist to carry both the IPv4 packet and the related metadata between the IPv4 Unicast LFB and IPv4 NextHop LFB across the two FEs.

The purpose of the inter-FE LFB is to define standard mechanisms for interconnecting FEs and for that reason we are not going to touch anymore on proprietary chip-chip interconnects other than state the fact they exist and that it is feasible to have translation to and from proprietary approaches. The focus is going to stick to FE-FE interconnect where the FE could be physical or virtual and the interconnecting technology runs a standard protocol such as ethernet, IP or other protocols on top of IP.

3.2. Arbitray Network Function

In this section we show an example of an arbitrary network function which is more coarse grained in terms of functionality. Each Network function may constitute more than one LFB.

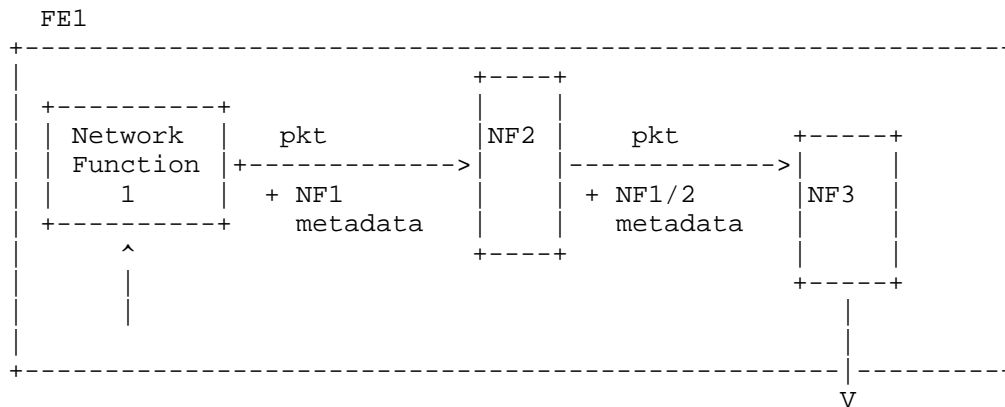


Figure 3: A Network Function Service Chain within one FE

The setup in Figure 3 is atypical of most packet processing boxes where we have functions like DPI, NAT, Routing, etc connected in such a topology to deliver a packet processing service to flows.

3.2.1. Distributing The Arbitray Network Function

The setup in Figure 3 can be split out across 3 FEs instead as demonstrated in Figure 4. This could be motivated by scale out reasons or because different vendors provide different functionality which is plugged-in to provide such functionality. The end result is to have the same packet service delivered to the different flows passing through.

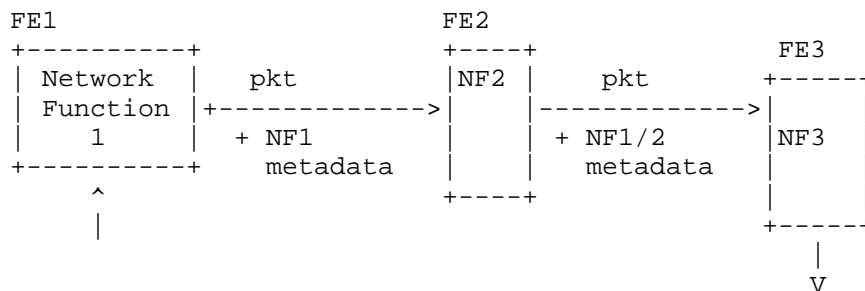


Figure 4: A Network Function Service Chain Distributed Across Multiple FEs

4. Proposal Overview

We address the inter-FE connectivity by proposing an inter-FE LFB. Using an LFB implies no change to the basic ForCES architecture in the form of the core LFBs (FE Protocol or Object LFBs). This design choice was made after considering an alternative approach that would have required changes to both the FE Object capabilities (SupportedLFBs) as well LFBTopology component to describe the inter-FE connectivity capabilities as well as runtime topology of the LFB instances.

4.1. Inserting The Inter-FE LFB

The distributed LFB topology described in Figure 2 is re-illustrated in Figure 5 to show the topology location where the inter-FE LFB would fit in.

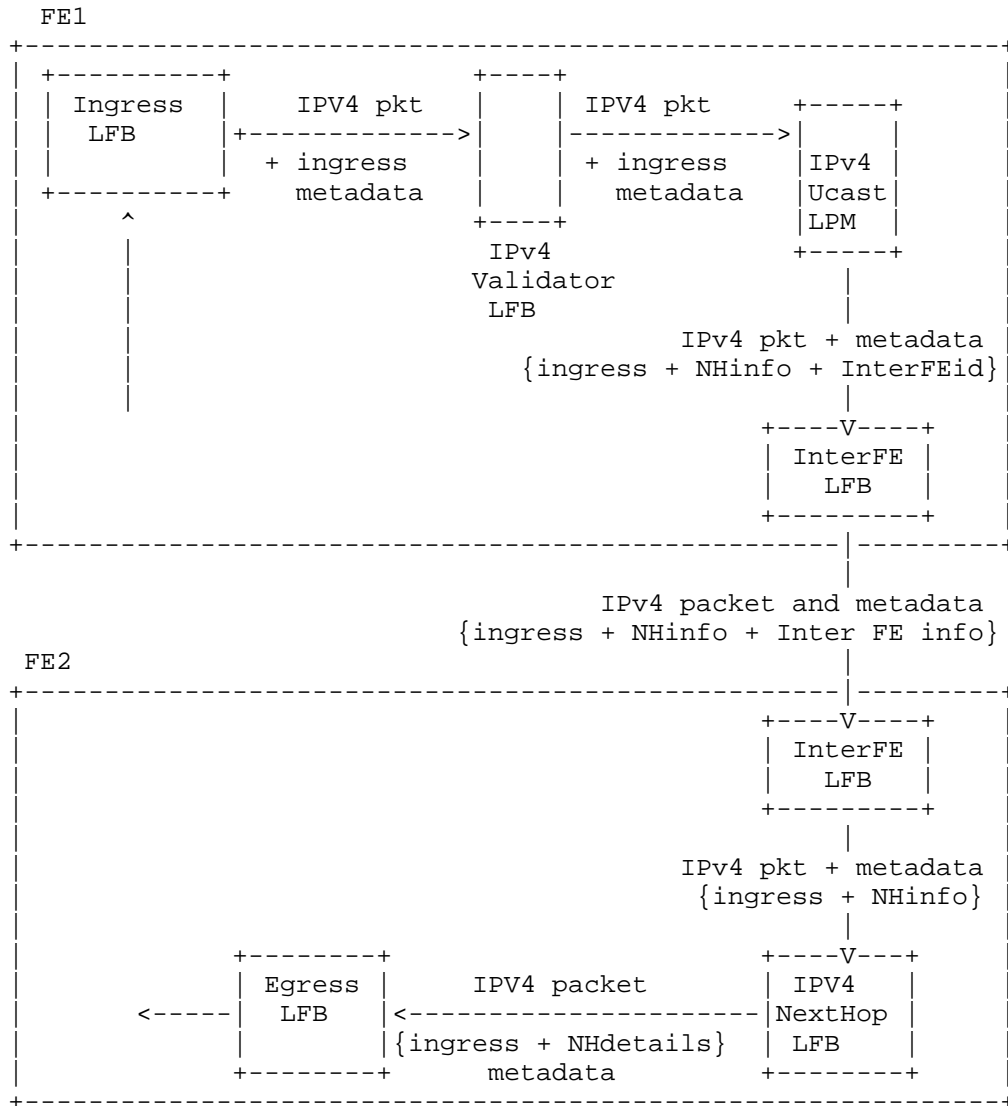


Figure 5: Split IPV4 forwarding service with Inter-FE LFB

As can be observed in Figure 5, the same details passed between IPV4 unicast LPM LFB and the IPV4 NH LFB are passed to the egress side of the Inter-FE LFB. In addition an index for the inter-FE LFB (interFEid) is passed as metadata.

The egress of the inter-FE LFB uses the received Inter-FE index (InterFEid metadata) to select details for encapsulation towards the

neighboring FE. These details will include what the source and destination FEID to be communicated to the neighboring FE. In addition the original metadata, any exception IDs may be passed along with the original IPV4 packet.

On the ingress side of the inter-FE LFB the received packet and its associated details are used to decide the graph continuation i.e which FE instance is to be passed the packet and what of the original metadata and exception IDs. In the illustrated case above, an IPV4 Nexthop LFB instance metadata is passed.

The ingress side of the inter-FE LFB consumes some of the information passed (eg the destination FEID) and passes on the IPV4 packet alongside with the ingress + NHinfo metadata to the IPV4 NextHop LFB as was done earlier in both Figure 1 and Figure 2.

4.2. Inter-FE connectivity

We describe the suggested encapsulation format (Figure 6) extended from the ForCES redirect packet format. We expect that for any transport mechanism used, that a description of how the different fields will be encapsulated to be explained. We provide a description of how ethernet encapsulation will be used in this case in Section 4.2.1.


```

+-- Main ForCES header
|
|   +---- msg type = REDIRECT
|   +---- Destination FEID
|   +---- Source FEID
|   +---- NEID (first word of Correlator)
|
+-- T = ExceptionID-TLV
|
|   +-- +-Exception Data ILV (I = exceptionID , L= length)
|   |
|   |   +----- V= Metadata value
|   |
|   |   .
|   |   .
|   |   .
|   |   +-Exception Data ILV
|   |
|   .
|
+-- T = METADATA-TLV
|
|   +-- +-Meta Data ILV (I = metaid, L= length)
|   |
|   |   +----- V= Metadata value
|   |
|   |   .
|   |   .
|   |   .
|   |   +-Meta Data ILV
|   |
|   .
+-- T = REDIRECTDATA-TLV
|
+-- Redirected packet Data

```

Figure 6: Packet format suggestion

- o The ForCES main header as described in RFC5810 is used as a fixed header to describe the Inter-FE encapsulation.
- * The Source ID field is mapped to the originating FE and the destination ID is mapped to the destination FEID.
- * The first 32 bits of the correlator field are used to carry the NEID. The 32-bit NEID defaults to 0.
- o The ExceptionID TLV carries one or more exception IDs within ILVs. The I in the ILV carries a globally defined exceptionID as per-ForCES specification defined by IANA. This TLV is new to ForCES and sits in the global ForCES TLV namespace.
- o The METADATA and REDIRECTDATA TLV encapsulations are taken directly from [RFC5810] section 7.9.

It is expected that a variety of transport encapsulations would be applicable to carry the format described in Figure 6. In such a case, a description of a mapping to interpret the inter-FE details and translate into proprietary or legacy formatting would need to be defined. For any mapping towards these definitions a different document to describe the mapping, one per transport, is expected to be defined.

4.2.1. Inter-FE Ethernet connectivity

In this specific document, we describe a format that is to be used over Ethernet.

The following describes the mapping from Figure 6 to ethernet wire encapsulation.

- o When an NE tag is needed, a VLAN tag will be used. Note: that the NEID as per Figure 6 is described as being 32 bits while a vlan tag is 12 bits. It is however thought to be sufficient to use 12 bits within the scope of a LAN cluster.
- o An ethernet type (To be defined) will be used to imply that a wire format is carrying an inter-FE LFB packet.
- o The destination FEID will be mapped to the destination MAC address of the target FEID.
- o The source FEID will be mapped to the source MAC address of the target FEID.
- o The ethernet data will carry the 3 defined TLVs on Figure 6 namely REDIRECTDATA TLV , METADATA TLV and ExceptionID TLV.

4.2.1.1. Inter-FE Ethernet Connectivity Issues

There are several issues that may arise due to using direct ethernet encapsulation.

- o The frame may end up being larger than the MTU. This is to be expected in particular where one LFB instance requires assembling for example a full IPV4 message before passing it downstream to another FE's LFB instance for further processing. There are several possible solutions:
 - * One possible solution is to use large MTUs; however, even that will have limits since the ethernet frames could grow arbitrarily large with increasing metadata being encapsulated.

- * An alternative approach is to add a fragmentation detail in the encapsulation. A simple approach is to have the inter-FE LFB (egress) add another header which submits total count of fragments and the fragment number of the submitted packet. The ingress of the inter-FE LFB will keep track of the fragments, assemble them as well as have a timer to discard outstanding fragments.
- * A third option is to limit the amount of metadata that could be transmitted so that the frame is sub-MTU size in presence of large MTU values. It will mean to add knobs to filter out or select which metadata gets encapsulated.
- * A fourth option is to use a transport that provides fragmentation services (such as IP).
- o The frame may be dropped if there is congestion on the receiving FE side. This may necessitate a retransmission mechanism to be built in. One approach to mitigate this issue is to make sure that inter-FE LFB frames receive the highest priority treatment when scheduled on the wire. A more common approach used in tunneling is to not care and let the packet originator to resend if they care about reliability. We do not expect to be any different.

Based on implementation experience and desire for simplicity, we choose to allow the control plane to decide how much metadata will be carried between FEs by setting the MTU sizes. An extra 64 bytes reserved on the MTU will account for 5 32-bit metadata or 3 64-bit metadata. In essence, the control plane making a decision for the MTU size is implicitly deciding how much metadata will be allowed.

5. Detailed Description of the inter-FE LFB

The inter-FE LFB has two LFB input ports and three LFB output ports.

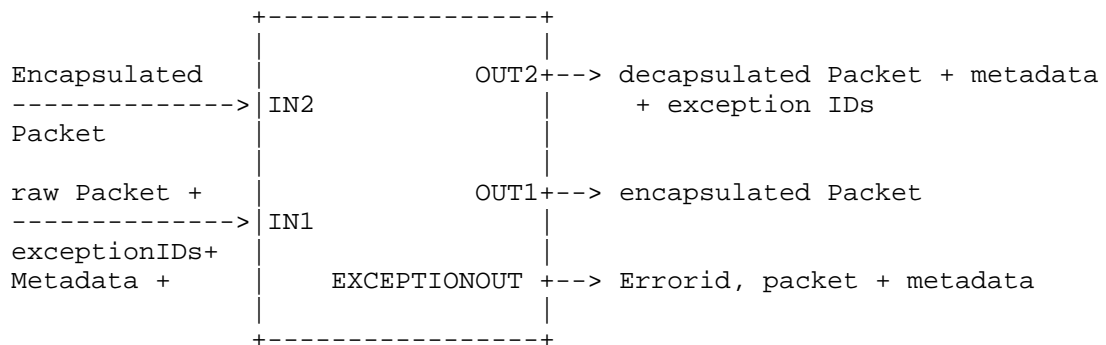


Figure 7: Inter-FE LFB

5.1. Data Handling

The Inter-FE LFB may be positioned at the egress of an FE. In such a case it receives via port IN1, raw packet, metadata, and exception IDs. The InterFEid metadatum MAY be present on the incoming raw data. The processed encapsulated packet will go out on either LFB port OUT1 to a downstream LFB or EXCEPTIONOUT port in the case of a failure.

The Inter-FE LFB may be positioned at the ingress of an FE. In such a case it receives, via port IN2, an encapsulated packet. Successful processing of the packet will result in a raw packet with associated metadata and exception IDs going downstream to an LFB connected on OUT2. On failure the data is sent out EXCEPTIONOUT.

An implementation may have one or more Ingress or egress inter-FE LFB instances. As an example, there could be one instance for the ingress side and a second instance for the egress side. An alternative approach maybe to have an ingress and egress instance per port.

The Inter-FE LFB uses the InterFEid metadatum when on an egress of an FE to lookup the NextFE table. The interFEid will be generated by an upstream LFB instance (i.e one preceeding the Inter-FE LFB). The output result constitutes a matched table row which has the InterFEinfo details i.e. the tuple {NEID, Destination FEID, Source FEID, metafilters, exceptionfilters}. The two filter lists define which Metadatum and/or exceptionids are to be passed to the neighboring FE. It is expected that zero configuration is needed; in the absence of the InterFEid metadatum, default behavior will be utilized.

5.1.1.1. Egress Processing

The InterFEid is used to lookup NextFE table. If lookup is successful, the inter-FE LFB will:

- o add the NEID data from the lookup result
- o walk the passed metadatum, apply the filters and encapsulate allowed ones them within METADATA-TLV as separate ILVs. The InterFEid is never passed.
- o walk all the passed exceptionIDs, apply the filters and encapsulate all allowed exception IDs within EXCEPTION-TLV header (as ILVs).
- o Encapsulate the data, if present, in REDIRECTDATA-TLV

The resulting packet is sent to the LFB instance connected to the OUT1 LFB port.

In the case of a failed lookup or a zero-value InterFEid, or absence of InterFEid, the default inter-FE LFB processing will:

- o Set the NEID to 0.
- o walk all the passed metadatum and encapsulate into the METADATA-TLV all metadatum. The InterFEid is never passed.
- o walk all the passed exceptionIDs and encapsulate each exceptionID within the EXCEPTION-TLV.
- o Encapsulate the data, if present, in REDIRECTDATA-TLV

The resulting packet is sent to the LFB instance connected to the OUT1 LFB port.

5.1.1.2. Ingress Processing

An inter-FE packet is recognized by looking at the ethertype.

In the ingress processing, the appropriate inter-FE LFB instance receives an encapsulated packet and extracts the packet data, metadata, and exception IDs. This data is then passed downstream to the next programmed LFB instance.

In the case of processing failure of either ingress or egress positioning of the LFB, the packet and metadata are sent out the EXCEPTIONOUT LFB port with proper error id (XXX: More description to

be added).

5.2. Metadata

A single (to be define from IANA space) metadatum, InterFEid, is defined.

5.3. Components

There is a single optional LFB component populated by the CE. The component is an array known as the NextFE table. Each row of the table constitutes the columns with {NEID, Destination FEID, Source FEID, array of allowed Metaids, array of allowed exception ids}. The table is looked up by a 32 bit index passed from an upstream LFB class instance in the form of InterFEid metadatum.

The CE programs LFB instances in a service graph that require inter-FE connectivity with InterFEid values to correspond to the inter-FE LFB NextFE table entries to use.

5.4. Inter-FE LFB XML Model

XXX: add ports and metadata definition

```
<LFBLibrary xmlns="urn:ietf:params:xml:ns:forces:lfbmodel:1.0"
  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  provides="IFE">
  <dataTypeDefs>

    <dataTypeDef>
      <name>IFEInfo</name>
      <synopsis>Describing IFE table row Information</synopsis>
      <struct>
        <component componentID="1">
          <name>NEID</name>
          <synopsis>the NEID</synopsis>
          <typeRef>uint32</typeRef>
        </component>
        <component componentID="2">
          <name>DST</name>
          <synopsis>the destination MAC address</synopsis>
          <typeRef>byte[6]</typeRef>
        </component>
        <component componentID="3">
          <name>SRC</name>
          <synopsis>the source MAC address</synopsis>
          <typeRef>byte[6]</typeRef>
        </component>
      </struct>
    </dataTypeDef>
  </dataTypeDefs>
```

```

    </component>
    <component componentID="4">
      <name>MetaFilterTab</name>
      <synopsis>the meta filter table</synopsis>
      <array type="variable-size">
        <typeRef>uint32</typeRef>
      </array>
    </component>
    <component componentID="5">
      <name>ExceptionFilterTab</name>
      <synopsis>the Exception filter table</synopsis>
      <array type="variable-size">
        <typeRef>uint32</typeRef>
      </array>
    </component>
  </struct>
</dataTypeDef>

</dataTypeDefs>

<LFBClassDefs>
  <LFBClassDef LFBClassID="6612">
    <name>IFE</name>
    <synopsis>
      This LFB describes IFE connectivity
    </synopsis>
    <version>1.0</version>

    <components>
      <component componentID="1" access="read-write">
        <name>NextFE</name>
        <synopsis>
          the table of all InterFE connectivities
        </synopsis>
        <array type="variable-size">
          <typeRef>IFEInfo</typeRef>
        </array>
      </component>
    </components>

  </LFBClassDef>
</LFBClassDefs>
</LFBLibrary>

```

Figure 8: Inter-FE LFB XML

6. Acknowledgements

The authors would like to thank Joel Halpern and Dave Hood for the stimulating discussions.

7. IANA Considerations

This memo includes one requests to IANA for InterFE Metaid.

8. Security Considerations

TBD

9. References

9.1. Normative References

- [RFC5810] Doria, A., Hadi Salim, J., Haas, R., Khosravi, H., Wang, W., Dong, L., Gopal, R., and J. Halpern, "Forwarding and Control Element Separation (ForCES) Protocol Specification", RFC 5810, March 2010.
- [RFC5812] Halpern, J. and J. Hadi Salim, "Forwarding and Control Element Separation (ForCES) Forwarding Element Model", RFC 5812, March 2010.

9.2. Informative References

- [RFC2119] Bradner, S., "Key words for use in RFCs to Indicate Requirement Levels", BCP 14, RFC 2119, March 1997.

Authors' Addresses

Damascene M. Joachimpillai
Verizon
60 Sylvan Rd
Waltham, Mass. 02451
USA

Email: damascene.joachimpillai@verizon.com

Jamal Hadi Salim
Mojatatu Networks
Suite 400, 303 Moodie Dr.
Ottawa, Ontario K2H 9R4
Canada

Email: hadi@mojatatu.com

ForCES WG
Internet-Draft
Intended status: Standards Track
Expires: July 18, 2015

B. Khasnabish
ZTE TX, Inc.
E. Haleplidis
University of Patras
J. Hadi Salim, Ed.
Mojatatu Networks
January 14, 2015

IETF ForCES Logical Function Block (LFB) Subsidiary Management
draft-khs-forces-lfb-subsidiary-management-05

Abstract

Deployment experience has demonstrated the value of using the Forwarding and Control Element Separation (ForCES) architecture to manage resources other than packet forwarding. In that spirit, the Forwarding Element Manager (FEM) is modelled by creating a Logical Functional Block (LFB) to represent its functionality. We refer to this LFB as the FE Configuration (FEC) LFB. A Control Element (CE) that controls a Forwarding Element's (FE) resources can also manage its configuration via the FEC LFB. This document introduces the FEC LFB, an LFB that specifies the configuration parameters of an FE.

Status of This Memo

This Internet-Draft is submitted in full conformance with the provisions of BCP 78 and BCP 79.

Internet-Drafts are working documents of the Internet Engineering Task Force (IETF). Note that other groups may also distribute working documents as Internet-Drafts. The list of current Internet-Drafts is at <http://datatracker.ietf.org/drafts/current/>.

Internet-Drafts are draft documents valid for a maximum of six months and may be updated, replaced, or obsoleted by other documents at any time. It is inappropriate to use Internet-Drafts as reference material or to cite them other than as "work in progress."

This Internet-Draft will expire on July 18, 2015.

Copyright Notice

Copyright (c) 2015 IETF Trust and the persons identified as the document authors. All rights reserved.

This document is subject to BCP 78 and the IETF Trust's Legal Provisions Relating to IETF Documents

(<http://trustee.ietf.org/license-info>) in effect on the date of publication of this document. Please review these documents carefully, as they describe your rights and restrictions with respect to this document. Code Components extracted from this document must include Simplified BSD License text as described in Section 4.e of the Trust Legal Provisions and are provided without warranty as described in the Simplified BSD License.

Table of Contents

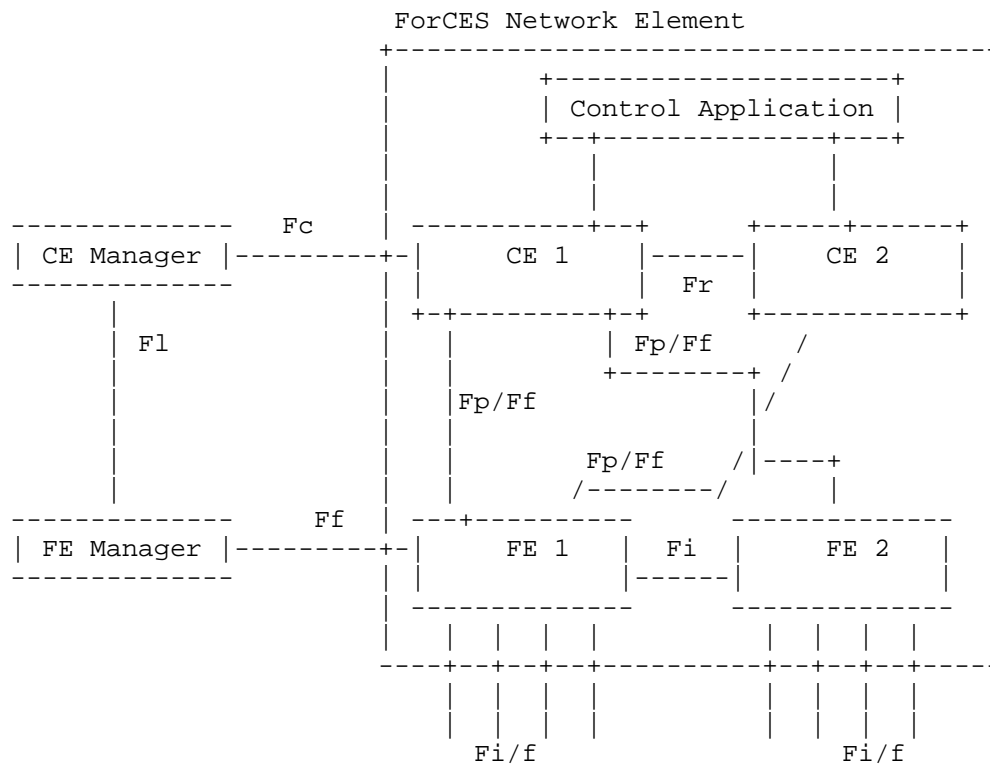
1. Introduction	3
1.1. Requirements Language	5
1.2. Definitions	5
2. Use cases	6
2.1. FE integration into an NE	6
2.2. CE associations	6
2.3. New LFB class installation	7
3. Applicability statement	7
3.1. FE Integrated	7
3.2. Virtual FEs	7
3.3. FEM	7
4. FEC Library	8
4.1. Frame Definitions	8
4.2. Datatype Definitions	8
4.3. Metadata Definitions	8
4.4. FEC	8
4.4.1. Data Handling	9
4.4.2. Components	9
4.4.3. Capabilities	9
4.4.4. Events	9
5. XML for FEC LFB	9
6. Security Considerations	13
7. IANA Considerations	13
7.1. LFB Class Names and LFB Class Identifiers	13
8. Acknowledgments	14
9. References	14
9.1. Normative References	14
9.2. Informative References	14
Appendix A. Appendix	15
A.1. Use of Virtualized ForCES Elements	15
A.1.1. Use of Virtualized CEs	15
A.1.2. Use of Virtualized FEs	15
A.1.3. Generic Lifecycle of Physical/Virtual Elements	15
A.2. Potential Scenarios	16
A.2.1. Recovery from FE failure	16
A.2.2. Recovery from CE failure	18
A.2.3. Load Balancing	19
A.2.4. Orchestration	19

A.2.5. Generic LFB Lifecycle Management	19
Authors' Addresses	19

1. Introduction

Deployment experience has demonstrated the value of using the Forwarding and Control Element Separation (ForCES) architecture to manage resources other than packet forwarding. In that spirit, the Forwarding Element Manager (FEM) is modelled by creating a Logical Functional Block (LFB) to represent its functionality. We refer to this LFB as the FE Configuration (FEC) LFB. A Control Element (CE) that controls a Forwarding Element's (FE) resources can also manage its configuration via the FEC LFB. This document introduces the FEC LFB, an LFB that specifies the configuration parameters of an FE.

On a running FE, a CE application may update an FE's runtime configuration via the FEC LFB.



Fp: CE-FE interface

Fc: Interface between the CE Manager and a CE

Ff: Interface between the FE Manager and an FE

Fl: Interface between the CE Manager and the FE Manager

Fi/f: FE external interface

Figure 1: ForCES Architectural Diagram

Figure 1 shows a control application manipulating, at runtime, FE config via the FEC LFB control. The above illustration is derived from Figure 1 in [RFC3746] with modifications showing the messaging for Ff (FEM to FE interface) going via the standard Fp plane. This is merely to demonstrate that the messaging is happening via the traditional Fp interface to the FEM/FEC; it does not however suggest moving away from the Ff interface.

The FEC LFB describes the configuration parameters of an FE, namely, the FEID, the FE IP address(es), the CEs it should be associated with, as well as the LFBs that it supports.

This document assumes that FEs are already booted. The FE's configuration can then be updated at runtime via the FEC LFB for runtime config purposes. This document does not specify or standardize the FEM-FE (Ff) interface as depicted in [RFC3746]. This document describes a mechanism with which a CE can instruct the FEC for FE management using ForCES.

In the case where we have a pool of unused packet processing resources that can be utilized as FEs, the FEC can be utilized to instruct the FE resource to join the Network Element cluster. The initiation would involve control of the creation, configuration, and resource assignment of FEs so as to be part of an NE. Appendix A describes how a resource pool of virtual machines could be initiated as basic CEs or FEs via an orchestration system and subsequently initiated into being part of an FE via the FEM. Again, it should be emphasized that the pools of FEs and CEs are already booted up by some resource owner, e.g. an FEM or an Element Management System (EMS). Subsidiary management provides the LFB library necessary, to manage and configure at runtime, these FEs to disconnect from the "resource pool CE" and join one or more CEs in the running NE.

This work item makes no assumption of whether FE resources are physical or virtual. In fact, the LFB library provided here is applicable to both. Thus it can also be useful in addressing control of virtual FEs where individual FEM Managers can be addressed to control the creation, configuration, and resource assignment of such virtual FEs within a physical FE.

1.1. Requirements Language

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 [RFC2119].

1.2. Definitions

This document follows the terminology defined by [RFC3654], [RFC3746], [RFC5810] and [RFC5812]. In particular, the reader is expected to be familiar with the following terms:

- o Logical Functional Block (LFB)
- o Forwarding Element (FE)
- o Control Element (CE)
- o ForCES Network Element (NE)

- o FE Manager (FEM)
- o CE Manager
- o ForCES Protocol
- o ForCES Protocol Layer (ForCES PL)
- o ForCES Protocol Transport Mapping Layer (ForCES TML)

2. Use cases

In this section we present sample use cases to illustrate the need and usefulness of the FEC LFB.

All use cases assume that an FEs and CEs have already been bootstrapped and instantiated but have not joined the NE. These FEs and CEs belong to a pool of untapped resources.

2.1. FE integration into an NE

The CE may request, for reasons such as performance, redundancy, load-sharing or new functionality request, to incorporate a new FE in the NE. The CE would be required to specify the following parameters. Firstly the FEID in order for the new FE to be uniquely identified within the NE. Second the FE IP address to bind to, IPv4 or IPv6. Thirdly the LFBs to be instantiated within the FE, by means of providing the LFB class, LFB version and LFB name. Finally the CEs that the new FE should associate within the NE as soon as it is integrated within the NE. This includes the CE IDs as well as their corresponding CE IP addresses.

2.2. CE associations

A CE may request for redundancy reasons that an FE to be associated to another CE as a backup at runtime. To achieve this goal, the master CE specifies the CEID of the new backup CE (to be uniquely identified within the NE) and the CE's IP address (IPv4 or IPv6, or IP addresses should the CE support multiple interfaces).

The CE will configure all FEC LFBs to the FEs within the NE of the CE ID and CE IP addresses in order for the FEs to perform the necessary actions ordered by the CE and described by [RFC7121].

the master CE, e.g. detecting a malfunctioning CE, could remove a backup CE from the FE.

2.3. New LFB class installation

A CE can learn via the capability of FEC LFB whether an FE is capable of loading new LFB classes. Provided that the FE supports new LFB class loading, the CE can request a new LFB to be installed and supported by the FE.

To load an LFB class on an FE, the CE will have to provide the LFB class and the LFB class version. There are optional fields which may be need to be described, depending on the implementation (out of scope for this document). Example:

location of the LFB Class to be installed and/or mechanism to download it. The exact detail of the location semantics is implementation specific and out of scope of this document.

Parameters needed by the LFB class module to allow for its initialization

3. Applicability statement

Examples of FEC usage are the following, but not limiting, three usage scenarios. These scenarios are not implementation details, but rather depict how the FEC class can be used to achieve the intended subsidiary mechanism for manipulating the configuration of FEs.

3.1. FE Integrated

Only one instance of the FEC class can exist and is directly related to the FE. The configuration parameters pertain to the parent FE.

3.2. Virtual FEs

In the case of the FE software that has hierarchical virtual FEs, multiple instances of the FEC class can exist, one per each virtual FE.

3.3. FEM

The third scenario, pertains to FEC LFB implementation as FE Manager paremeters. In such a case, the FE configurations are locally and logically centralized by the FE manager. The FE manager may hold multiple instances of the FEC class in the FEM, one per each FE. Using the ForCES protocol a CE, through the Fp interface, or a CE Manager via the Fl interface, will instruct the FEM to change the configuration of the FEs. The FEM may hold more information pertaining the NE, such as the topology and chaining of FEs which the

CE would require to alter, along with the FE changes. In such a case the Ff interface is out of scope.

4. FEC Library

4.1. Frame Definitions

This LFB does not define any frames

4.2. Datatype Definitions

This library defines the following datatypes.

Data Type Name	Type	Synopsis
IPs	A Struct of 2 components. IPv4 (byte[4]) and IPv6 (byte[16]) addresses.	A struct that defines an IPv4 and an IPv6 address
LFBDefs	A Struct that contains three components. The LFB Class ID (uint32), the LFB version (string) and optional the LFB name (string) and the location of the LFB where it will be retrieved from.	A struct that defines basic LFB definitions
CEParams	A Struct that contains two components. A CE's ID (uint32) and the CE's IPs (array of IPs)	A struct that defines CE parameters.

FEM Data Types

4.3. Metadata Definitions

This LFB does not define any metadata definition

4.4. FEC

The FE Configuration LFB is an LFB that standardizes configuration of the FE parameters.

4.4.1. Data Handling

The FEC LFB does not handle any packets. It's function is to provide the configuration parameters to the CE to be updated at runtime.

4.4.2. Components

This LFB has four components specified. The FEID, a uint32 component that defines the ID of the FE. The FEIP, a table of IP address, and each row is a struct of an IPv4 and an IPv6 address. The LFB Parameters, a table of LFBs, each row a struct of LFB Class ID, LFB Version and optional LFB name and location. Finally the CEs, a table of CE parameters, each row a struct of a CE ID and a table of CE IPs.

4.4.3. Capabilities

This capability specifies whether this FE supports dynamic loading of new LFBs.

4.4.4. Events

This LFB has five events specified. These events notify the CE whether the FEID has been changed, an entry of the FEIP table has been created or changed and an entry of the CE information added or deleted. The event reports are the respective data that has been modified.

5. XML for FEC LFB

```
<?xml version="1.0" encoding="UTF-8"?>
<LFBLibrary xmlns="urn:ietf:params:xml:ns:forces:lfbmodel:1.0"
  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance" provides="FEC">
  <dataTypeDefs>
    <dataTypeDef>
      <name>IPs</name>
      <synopsis>IP definition</synopsis>
      <struct>
        <component componentID="1">
          <name>FEIPv4</name>
          <synopsis>The FEs IPv4</synopsis>
          <typeRef>byte[4]</typeRef>
        </component>
        <component componentID="2">
          <name>FEIPv6</name>
          <synopsis>The FEs IPv6</synopsis>
          <typeRef>byte[16]</typeRef>
        </component>
      </struct>
    </dataTypeDef>
  </dataTypeDefs>
```

```
</dataTypeDef>
<dataTypeDef>
  <name>LFBDefs</name>
  <synopsis>LFB parameters inside the FE</synopsis>
  <struct>
    <component componentID="1">
      <name>LFBClassID</name>
      <synopsis>The LFB Class ID</synopsis>
      <typeRef>uint32</typeRef>
    </component>
    <component componentID="2">
      <name>LFBVersion</name>
      <synopsis>The Version of the LFB</synopsis>
      <typeRef>string</typeRef>
    </component>
    <component componentID="3">
      <name>LFBName</name>
      <synopsis>The name of the LFB</synopsis>
      <optional/>
      <typeRef>string</typeRef>
    </component>
    <component componentID="4">
      <name>LFBLocation</name>
      <synopsis>The location of the LFB to be retrieved
        from</synopsis>
      <optional/>
      <typeRef>string</typeRef>
    </component>
  </struct>
</dataTypeDef>
<dataTypeDef>
  <name>CEParams</name>
  <synopsis>CE parameters</synopsis>
  <struct>
    <component componentID="1">
      <name>CEID</name>
      <synopsis>The CE ID</synopsis>
      <typeRef>uint32</typeRef>
    </component>
    <component componentID="2">
      <name>CEIP</name>
      <synopsis>The CEIP</synopsis>
      <array>
        <typeRef>IPs</typeRef>
      </array>
    </component>
  </struct>
</dataTypeDef>
```

```
</dataTypeDefs>
<LFBClassDefs>
  <LFBClassDef LFBClassID="21">
    <name>FEC</name>
    <synopsis>Forwarding Element Configuration</synopsis>
    <version>1.0</version>
    <components>
      <component componentID="1" access="read-write">
        <name>FEID</name>
        <synopsis>The FEID</synopsis>
        <typeRef>uint32</typeRef>
      </component>
      <component componentID="2" access="read-write">
        <name>FEIP</name>
        <synopsis>The FE's IP</synopsis>
        <array>
          <typeRef>IPs</typeRef>
        </array>
      </component>
      <component componentID="3" access="read-write">
        <name>LFBparameters</name>
        <synopsis>The LFBs in this FE</synopsis>
        <array>
          <typeRef>LFBDefs</typeRef>
        </array>
      </component>
      <component componentID="4" access="read-write">
        <name>CEs</name>
        <synopsis>The CEs that should be associated with this
          FE</synopsis>
        <array>
          <typeRef>CEParams</typeRef>
        </array>
      </component>
    </components>
    <capabilities>
      <capability componentID="10">
        <name>DynamicLFBLoading</name>
        <synopsis>This capability specifies whether this FE supports
          dynamic loading of new LFBs</synopsis>
        <typeRef>boolean</typeRef>
      </capability>
    </capabilities>
    <events baseID="20">
      <event eventID="1">
        <name>IDChanged</name>
        <synopsis>The FE ID has been changed</synopsis>
        <eventTarget>
```

```
        <eventField>FEID</eventField>
    </eventTarget>
    <eventChanged/>
    <eventReports>
        <eventReport>
            <eventField>FEID</eventField>
        </eventReport>
    </eventReports>
</event>
<event eventID="2">
    <name>FEIPChanged</name>
    <synopsis>An IP of the FE has been changed</synopsis>
    <eventTarget>
        <eventField>FEIP</eventField>
    </eventTarget>
    <eventCreated/>
    <eventReports>
        <eventReport>
            <eventField>FEIP</eventField>
            <eventSubscript>_FEIPsrowid_</eventSubscript>
        </eventReport>
    </eventReports>
</event>
<event eventID="3">
    <name>FEIPCreated</name>
    <synopsis>An FEIP has been deleted</synopsis>
    <eventTarget>
        <eventField>FEIP</eventField>
    </eventTarget>
    <eventDeleted/>
    <eventReports>
        <eventReport>
            <eventField>FEIP</eventField>
            <eventSubscript>_FEIPsrowid_</eventSubscript>
        </eventReport>
    </eventReports>
</event>
<event eventID="4">
    <name>CEAdded</name>
    <synopsis>An CE has been added</synopsis>
    <eventTarget>
        <eventField>CEs</eventField>
    </eventTarget>
    <eventCreated/>
    <eventReports>
        <eventReport>
            <eventField>CEs</eventField>
        </eventReport>
    </eventReports>
</event>
```

```
        </eventReports>
    </event>
    <event eventID="5">
        <name>CEDeleted</name>
        <synopsis>An CE has been added</synopsis>
        <eventTarget>
            <eventField>CEs</eventField>
        </eventTarget>
        <eventDeleted/>
        <eventReports>
            <eventReport>
                <eventField>CEs</eventField>
            </eventReport>
        </eventReports>
    </event>
</events>
</LFBClassDef>
</LFBClassDefs>
</LFBLibrary>
```

Figure 2: FEM XML LFB library

6. Security Considerations

Security considerations for ForCES LFB subsidiary management will be added in a future version of this draft.

7. IANA Considerations

7.1. LFB Class Names and LFB Class Identifiers

LFB classes defined by this document belong to LFBs defined by Standards Track RFCs. According to IANA, the registration procedure is Standards Action for the range 0 to 65535 and First Come First Served with any publicly available specification for over 65535. This specification includes the following LFB class names and LFB class identifiers:

LFB Class Identifier	LFB Class Name	LFB Version	Description	Reference
21	FEC	1.0	An FEC LFB to standardize creation of ForCES Network Elements	This document

Logical Functional Block (LFB) Class Names and Class Identifiers

8. Acknowledgments

The authors would like to thank DJ, Joel, ChuanhuangLi, and many others for their discussions and support.

9. References

9.1. Normative References

- [RFC5810] Doria, A., Hadi Salim, J., Haas, R., Khosravi, H., Wang, W., Dong, L., Gopal, R., and J. Halpern, "Forwarding and Control Element Separation (ForCES) Protocol Specification", RFC 5810, March 2010.
- [RFC5812] Halpern, J. and J. Hadi Salim, "Forwarding and Control Element Separation (ForCES) Forwarding Element Model", RFC 5812, March 2010.
- [RFC7121] Ogawa, K., Wang, W., Haleplidis, E., and J. Hadi Salim, "High Availability within a Forwarding and Control Element Separation (ForCES) Network Element", RFC 7121, February 2014.

9.2. Informative References

- [RFC2119] Bradner, S., "Key words for use in RFCs to Indicate Requirement Levels", BCP 14, RFC 2119, March 1997.
- [RFC3654] Khosravi, H. and T. Anderson, "Requirements for Separation of IP Control and Forwarding", RFC 3654, November 2003.
- [RFC3746] Yang, L., Dantu, R., Anderson, T., and R. Gopal, "Forwarding and Control Element Separation (ForCES) Framework", RFC 3746, April 2004.

Appendix A. Appendix

A.1. Use of Virtualized ForCES Elements

Virtualization of ForCES Elements allows efficient, scalable, and robust utilization of network control and transmission resources. Virtualization has been discussed (and deployed) widely in the Computing Industry (e.g., server) in the context of efficient utilization of server resources.

As mentioned before, the currently existing techniques and solutions may be either slow or not directly applicable to ForCES LFB subsidiary management.

A.1.1. Use of Virtualized CEs

In this section we discuss the use of virtualized ForCES control elements (CEs). The resulting operating entities in virtualized environment are Virtual CEs or VCEs. The CE Visor (CEV) has the visibility to all of the VCEs in a domain, and can assign one of the VCEs as primary Master-VCE and another as secondary Master-VCE. CEV can dynamically manage the role of primary and secondary master-VCEs from a pool of VCEs.

A.1.2. Use of Virtualized FEs

In this section we discuss the use of virtualized ForCES forwarding elements (FEs). The resulting operating entities in virtualized environment are Virtual FEs or VFES. The FE Visor (FEV) has the visibility to all of the VFES in a domain, and can assign one of the VFES as primary Master-VFE and another as secondary Master-VFE. FEV can dynamically manage the role of primary and secondary master-VFES from a pool of VFES.

A.1.3. Generic Lifecycle of Physical/Virtual Elements

The generic lifecycle of physical/virtual elements including NEs, FEs, VNEs, VCEs, VFES, etc. consists of the following FOUR states:

- o (a) Instantiation -- This refers to instantiation of CEs and FEs.
- o (b) Association -- This refers to associating FEs to the CEs
- o (c) Activation -- This refers to activation of CEs and FEs for normal operation. This state may include monitoring as well with an objective to satisfy both scaling and reliability requirements.

- o (d) Release -- This refers to releasing resources (both physical and virtual elements) to the pool of available (that is un-assigned) elements, and reporting this to the appropriate (CE or FE) manager. It may be required to cleanse the physical/virtual elements before releasing in order to prevent harvesting of data/information by the the next user of the CEs/FEs. The details of the cleansing operation is out of scope of this draft.

Figure 1 shows physical/virtual elements states and their transition.

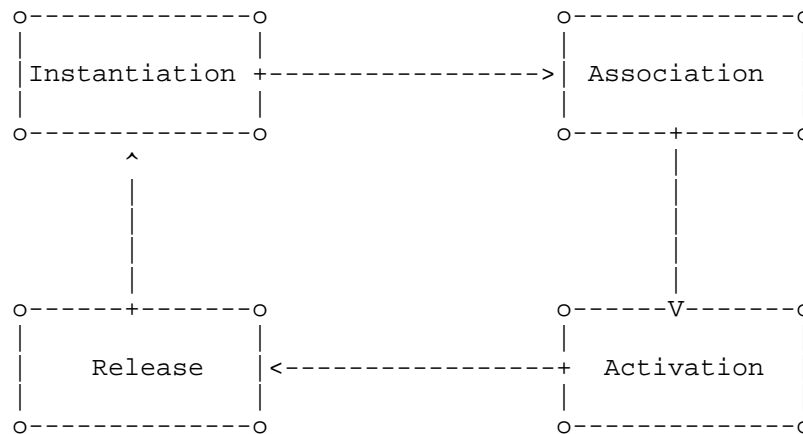


Figure 1: Physical/Virtual Elements States and their Transition

A.2. Potential Scenarios

In this section we discuss a few potential scenarios that can utilize ForCES LFB subsidiary management for efficient and robust operation of networks without using excessive additional resources.

A.2.1. Recovery from FE failure

In this section we discuss how virtualization of FEs can be used for efficient recovery from FE failure(s). An FE can initially boot using a default Association and Configuration. The Association and Configuration can be updated at runtime via an FE-Visor or FEC LFB for runtime configuration purposes. This can be achieved, for example, by adding a new CE and its associated IP address. A CE can initially boot using a default Configuration and State(s). The Configuration and State(s) can be updated at runtime via a CE-Visor or a similar CE Configuration (CEC) LFB to satisfy the runtime requirements.

- o Step-5: The Controller (attached to FEz) instructs FEz to increase its Syslog debug level. This is essentially the "Configuration" part of Association and Configuration, as discussed earlier.

Note that the 4th (FEM part of the charter) and 5th steps are what we would like to achieve here. In addition, the FEVM may not need to be aware which Virtual FEs are in one Virtual NE, it only needs know of the information about a Virtual FE in the physical FE. CE Manager may need to have visibility to all Virtual NEs. The component "NE" of the LFB may be considered as Virtual NE as well.

A.2.2. Recovery from CE failure

In this section we discuss how virtualization of CEs can be used for efficient recovery from CE failure(s).

A CE can initially boot using a default Association, State, and Configuration. The Association and Configuration can be updated at runtime via a CE-Visor or FEC-LFB for runtime configuration purposes, for example, by adding a new CE and its associated IP address.

An FE can initially boot using a default Configuration, Association, with a CE, and State. The Configuration, Association can be updated at runtime via a FE-Visor or FEC LFB to satisfy runtime requirements. The sequence of events, an example, can be as follows.

- o Step-1: The CEx is Active with CEy as its Standby with Standby-Active or Active-Active setup.
- o Step-2: The CEx controls FEy and FEw with both FEy and FEw having Standby control links to CEy, with Standby-Active or Active-Active setup. Note that CEx and CEy are controlled, assigned, by CE-visor, and may have a common, virtual, IP address.
- o Step-3: The Controller is fully aware of the status of all of the CEs, physical and virtual; When CEx fails, its states are fully transferred (may already be synced) to CEy.
- o Step-4: The Standby links from CEy to FEy and FEw become fully active, and the control, of FEy and FEw, is fully transferred from CEx and CEy.
- o Step-5: A graceful-smooth failover of CEx to CEy is now successfully complete, and SysLog debug level for CEy is increased..

As discussed earlier, the last two steps are concerned with Subsidiary management. Although we discuss the recovery method by

using virtualization of CEs, the role of FEVM in the recovery process will be described further later.

A.2.3. Load Balancing

In this section we discuss efficient load balancing of both CE and FE in virtualized environment.

A.2.4. Orchestration

In this section we discuss efficient Orchestration of both CE and FE in virtualized multi-admin-domain environment.

A.2.5. Generic LFB Lifecycle Management

In this section we discuss generic lifecycle management of subsidiaries of LFBs in virtualized environment(s). The typical management activities in the life of FE/CE are discussed in the following sub-sections.

A.2.5.1. Booting a CE/FE

When an entity needs to boot a CE/FE, if this is a VM, some orchestration would scheme/plan to do this. In case of ForCES, we have a control App that boots a CE or an FE via a management FE. So here we have a management plane details that is described either in FEM or other LFB.

A.2.5.2. Bootstrapping the Configuration

The FE, e.g., the VM which has just been booted, as described in the previous sub-section, needs initial bootstrap configuration (e.g., what CEs to connect to etc). This clearly falls in the FEC LFB domain.

A.2.5.3. Runtime Management

At runtime of the FE, for example, the management could introduce a new CE for the FE to associate with; it may also be for an FE to dissociate from a CE, and so on.

Authors' Addresses

Bhumip Khasnabish
ZTE TX, Inc.
55 Madison Avenue, Suite 160
Morristown, New Jersey 07960
USA

Phone: +001-781-752-8003
Email: vumip1@gmail.com, bhumip.khasnabish@ztetx.com
URI: <http://tinyurl.com/bhumip/>

Evangelos Haleplidis
University of Patras
Department of Electrical and Computer Engineering
Patras 26500
Greece

Email: ehalep@ece.upatras.gr

Jamal Hadi Salim (editor)
Mojatatu Networks
Suite 400, 303 Moodie Dr.
Ottawa, Ontario K2H 9R4
Canada

Email: hadi@mojatatu.com