Internet Engineering Task Force (IETF)
Request for Comments: 7409
Category: Experimental
ISSN: 2070-1721

E. Haleplidis
University of Patras
J. Halpern
Ericsson

November 2014

Forwarding and Control Element Separation (ForCES) Packet Parallelization

Abstract

Many network devices support parallel packet processing. This document describes how Forwarding and Control Element Separation (ForCES) can model a network device's parallelization datapath using constructs defined by the ForCES model (RFC 5812) and controlled via the ForCES protocol (RFC 5810).

Status of This Memo

This document is not an Internet Standards Track specification; it is published for examination, experimental implementation, and evaluation.

This document defines an Experimental Protocol for the Internet community. This document is a product of the Internet Engineering Task Force (IETF). It represents the consensus of the IETF community. It has received public review and has been approved for publication by the Internet Engineering Steering Group (IESG). Not all documents approved by the IESG are a candidate for any level of Internet Standard; see Section 2 of RFC 5741.

Information about the current status of this document, any errata, and how to provide feedback on it may be obtained at http://www.rfc-editor.org/info/rfc7409.

Table of Contents

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 in development since 1994 at the Massachusetts Institute of Technology (MIT) Laboratory. Cilk allows programmers to identify elements that can be executed in parallel. The two Cilk concepts used in this document are "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 (see Section 1.2 for the definitions of both "task" and "task correclator").

This document is Experimental; 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 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 belonging 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 into chunks of equal size 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 physically copy the packet and send all packets on the parallel paths or decide to logically copy the packet by simply sending, for example, pointers to 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 be 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 before and after the parallelization occurs:

1. Splitter: Similar to Cilk's spawn, a splitter is an LFB that will split the path of a packet that will be sent to multiple downstream LFBs to be processed in parallel.

2. Merger: Similar to Cilk's sync, a merger is 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. The example in Figure 1 depicts multiple regular expression (regex) match LFBs that perform match operations on parts of the original packet. Figure 2 shows an example of a flood parallel datapath along with the Splitter and Merger LFB. The example in Figure 2 depicts a path that will classify an IPv4 packet while also performing metering; on the other path, the IPv4 Time to Live (TTL) field will be decremented.

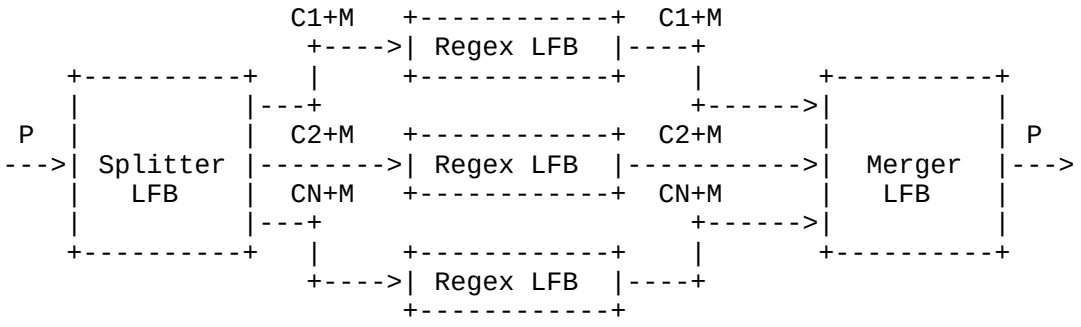

Figure 1: Simple Split Parallel Processing

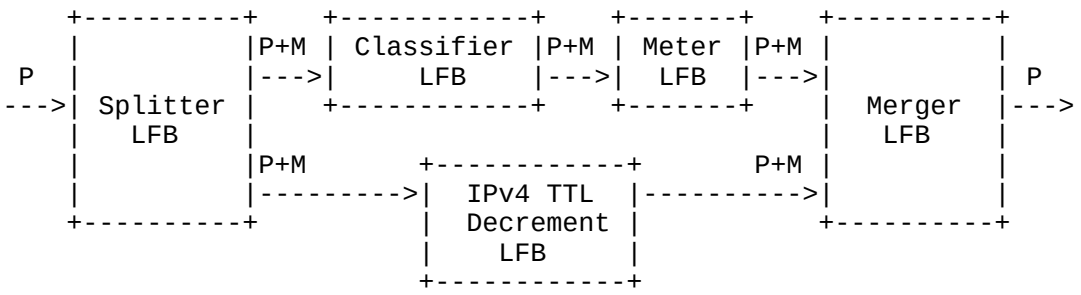

Figure 2: Simple Flood Parallel Processing

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 is required 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 need to be defined, as well as an augmentation to the defined metadata.

2.1. CoreParallelization 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:

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

DataType 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 [RFC7408], 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 chunk 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.

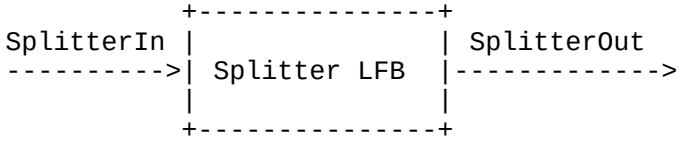

Figure 4: Splitter LFB

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 instances of the group output "SplitterOut". If the parallel type is of type split (1), then the packet will be split into same size chunks except for 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 instances of the group output "ParallelOut" or in any other way defined by the implementer. Each packet or chunk will be accompanied by the following metadata set as a struct:

- ParallelType - The parallel type: split or flood.
- ParallelID - Generated by the Splitter LFB to identify which chunks or packets belong to the same parallel task.
- 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.
- 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, a uint32 that defines how the packet will be processed by the Splitter LFB. The second is the ChunkSize, a 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 is a struct of two uint32s 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 1 and Figure 2.

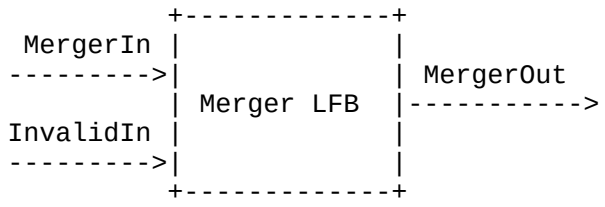

Figure 5: Merger LFB

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, the TaskCorrelator, the ParallelNum, and the ParallelPartsCount.

In the case in which an upstream LFB has dropped a packet or a chunk, the Merger LFB MAY receive only the metadata, both the metadata and the packet, or the chunk through the InvalidIn group input port. It SHOULD receive a metadata specifying the error code. Currently defined metadata 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 it 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 be dropped or 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 have 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 the 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 events. 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 can 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>
      <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 to output instances.  Each chunk is sent only
                  once by the LFB.</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 chunks of the same size</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, it 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 the 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 that 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. IANA Considerations

6.1. LFB Class Names and LFB Class Identifiers

LFB classes defined by this document do not belong to LFBs defined by Standards Action. As such, the corresponding values assigned in the "Logical Functional Block (LFB) Class Names and Class Identifiers" registry at <http://www.iana.org/assignments/forces> are above 65535.

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

LFB Class ID	LFB Class Name	LFB Version	Description	Ref
65537	Ext-Splitter	1.0	A Splitter LFB will send either the same packet or chunks of one packet to multiple LFBs.	RFC 7409
65538	Ext-Merger	1.0	A Merger LFB receives multiple packets or multiple chunks of the same packet and merges them into one.	RFC 7409
65539	Ext-CoreParallelization	1.0	A core LFB to signify the parallelization capability	RFC 7409

Logical Functional Block (LFB) Class Names and Class Identifiers

6.2. Metadata ID

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

Value	Name	Reference
0x00000010	ParallelMetadataSet	RFC 7409

Metadata ID Assigned by this Specification

7. Security Considerations

This document does not alter either the ForCES model [RFC5812] or the ForCES protocol [RFC5810]. As such, it 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.

8. References

8.1. Normative References

[RFC2119] Bradner, S., "Key words for use in RFCs to Indicate Requirement Levels", BCP 14, RFC 2119, March 1997, <http://www.rfc-editor.org/info/rfc2119>.

[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, <http://www.rfc-editor.org/info/rfc5810>.

[RFC5812] Halpern, J. and J. Hadi Salim, "Forwarding and Control Element Separation (ForCES) Forwarding Element Model", RFC 5812, March 2010, <http://www.rfc-editor.org/info/rfc5812>.

[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, <http://www.rfc-editor.org/info/rfc6956>.

[RFC7408] Haleplidis, E., "Forwarding and Control Element Separation (ForCES) Model Extension", RFC 7408, November 2014, <http://www.rfc-editor.org/info/rfc7408>.

8.2. Informative References

[Cilk] Massachusetts Institute of Technology, "The Cilk Project", <http://supertech.csail.mit.edu/cilk/>.

Acknowledgments

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

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
United States

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