

Internet Engineering Task Force
Internet-Draft
Intended status: Informational
Expires: April 20, 2013

E. Haleplidis
University of Patras
October 17, 2012

ForCES Model Extension
draft-haleplidis-forces-model-extension-01

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, so that control elements (CEs) can control the FEs accordingly. More specifically, the model describes the logical functions that are present in an FE, what capabilities these functions support, and how these functions are or can be interconnected.

RFC5812 has been around for two years and experience in its use has shown room for extensions without a need to alter the protocol while retaining backward compatibility with older xml libraries.

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 20, 2013.

Copyright Notice

Copyright (c) 2012 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	5
3. ForCES Model Extension overview	6
4. Extensions	7
4.1. Complex Metadata	7
4.2. DataType Optional Default Value	9
4.3. Enhancing XML Validation	9
5. XML Extension Schema for LFB Class Library Documents	11
6. Acknowledgements	24
7. IANA Considerations	25
8. Security Considerations	26
9. References	27
9.1. Normative References	27
9.2. Informative References	27
Author's Address	28

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.

Element - Element is generally used in this document in accordance with the XML usage of the term. It refers to an XML tagged part of an XML document. For a precise definition, please see the full set of XML specifications from the W3C. This term is included in

this list for completeness because the ForCES formal model uses XML.

Attribute - Attribute is used in the ForCES formal modeling in accordance with standard XML usage of the term, i.e., to provide attribute information included in an XML tag.

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 Class Library - The LFB class library is a set of LFB classes that has been identified as the most common functions found in most FEs and hence should be defined first by the ForCES Working Group.

2. Introduction

The ForCES Model [RFC5812] presents a formal way to define FEs Logical Function Blocks (LFBs) using XML. [RFC5812] has been published a little more than two years and current experience in its use has shown some room for adding new and changing existing modeling concepts.

This document extends the ForCES Model by changing and adding new concepts. These extensions do not require any changes on the ForCES protocol [RFC5810] as they are simply changes of the schema definition. Additionally backward compatibility is ensured as xml libraries produced with the earlier schema are still valid with the new one.

3. ForCES Model Extension overview

The following extensions are considered:

1. Allow complex metadata.
2. Allow optional default values for datatypes.

Additionally this document is also enhancing the XML validation.

4. Extensions

Some of these extensions were product of the work done on the OpenFlow library [I-D.haleplidis-forces-openflow-lib] document.

4.1. Complex Metadata

Section 4.6. (Element for Metadata Definitions) in the ForCES Model [RFC5812] limits the datatype use in metadata to only atomic types. Figure 1 shows the xml schema excerpt where only typeRef and atomic are allowed for a metadata definition.

With this extension (Figure 2), complex data types are also allowed, specifically structs and arrays as metadata. The key declarations are required to check for validity of content keys in arrays and componentIDs in structs.

```
<xsd:complexType name="metadataDefsType">
  <xsd:sequence>
    <xsd:element name="metadataDef" maxOccurs="unbounded">
      <xsd:complexType>
        <xsd:sequence>
          <xsd:element name="name" type="xsd:NMTOKEN"/>
          <xsd:element ref="synopsis"/>
          <xsd:element name="metadataID" type="xsd:integer"/>
          <xsd:element ref="description" minOccurs="0"/>
          <xsd:choice>
            <xsd:element name="typeRef" type="typeRefNMTOKEN"/>
            <xsd:element name="atomic" type="atomicType"/>
          </xsd:choice>
        </xsd:sequence>
      </xsd:complexType>
    </xsd:element>
  </xsd:sequence>
</xsd:complexType>
```

Figure 1: Initial MetadataDefType Defintion in the schema

```

<xsd:complexType name="metadataDefsType">
  <xsd:sequence>
    <xsd:element name="metadataDef" maxOccurs="unbounded">
      <xsd:complexType>
        <xsd:sequence>
          <xsd:element name="name" type="xsd:NMTOKEN"/>
          <xsd:element ref="synopsis"/>
          <xsd:element name="metadataID" type="xsd:integer"/>
          <xsd:element ref="description" minOccurs="0"/>
          <xsd:choice>
            <xsd:element name="typeRef" type="typeRefNMTOKEN"/>
            <xsd:element name="atomic" type="atomicType"/>
            <xsd:element name="array" type="arrayType">
              <xsd:key name="contentKeyID1">
                <xsd:selector xpath="lfb:contentKey"/>
                <xsd:field xpath="@contentKeyID"/>
              </xsd:key>
            </xsd:element>
            <xsd:element name="struct" type="structType">
              <xsd:key name="structComponentID1">
                <xsd:selector xpath="lfb:component"/>
                <xsd:field xpath="@componentID"/>
              </xsd:key>
            </xsd:element>
          </xsd:choice>
        </xsd:sequence>
      </xsd:complexType>
    </xsd:element>
  </xsd:sequence>
</xsd:complexType>

```

Figure 2: New MetadataDefType Definition for the schema

Two simple use cases can be seen in the OpenFlow switch [OpenFlowSpec1.1]:

1. The Action Set metadata follows a packet inside the Flow Tables. The Action Set metadata is an array of actions to be performed at the end of the pipeline.
2. When a packet is received from a controller it may be accompanied by a list of actions to be performed on it prior to be sent on the flow table pipeline which is also an array.

4.2. DataType Optional Default Value

In the original schema, default values can be defined only in datatypes defined inside LFB components. However when it's a complex datatype or it's a referred datatype, using the default value field is difficult to be used. Additionally there is no option in a complex datatype to use the default value field for only one component in the complex datatype.

This extension allows optionally to add default values to atomic and typeref types, whether they are as simple or complex datatypes. A simple use case would be to have a struct component where one of the components is a counter which the default value would be zero.

This extension alters the definition of the typeDeclarationGroup in the xml schema from Figure 3 to Figure 4 to allow default values to TypeRef.

```
<xsd:element name="typeRef" type="typeRefNMTOKEN"/>
```

Figure 3: Initial Excerpt of typeDeclarationGroup Definition in the schema

```
<xsd:sequence>
  <xsd:element name="typeRef" type="typeRefNMTOKEN"/>
  <xsd:element name="DefaultValue" type="xsd:token"
    minOccurs="0"/>
</xsd:sequence>
```

Figure 4: New Excerpt of typeDeclarationGroup Definition in the schema

Additionally it appends to the declaration of the AtomicType this xml (Figure 5) to allow default values to Atomic datatypes.

```
<xsd:element name="defaultValue" type="xsd:token" minOccurs="0"/>
```

Figure 5: Appending xml in of AtomicType Definition in the schema

4.3. Enhancing XML Validation

As specified earlier this is not an extension but an enhancement of the schema to provide additional validation rules. This includes adding new key declarations to provide uniqueness as defined by the ForCES Model [RFC5812]. Such validations work only on within the same xml file.

The following validation rules have been appended in the original

schema in [RFC5812]:

1. Each metadata ID must be unique.
2. LFB Class IDs must be unique.
3. Component ID, Capability ID and Event Base ID must be unique per LFB.
4. Event IDs must be unique per LFB.
5. Special Values in Atomic datatypes must be unique per atomic datatype.

5. XML Extension Schema for LFB Class Library Documents

```
<?xml version="1.0" encoding="UTF-8"?>
<xsd:schema xmlns:xsd="http://www.w3.org/2001/XMLSchema"
  xmlns="urn:ietf:params:xml:ns:forces:lfbmodel:1.0"
  xmlns:lfb="urn:ietf:params:xml:ns:forces:lfbmodel:1.0"
  targetNamespace="urn:ietf:params:xml:ns:forces:lfbmodel:1.0"
  elementFormDefault="qualified" attributeFormDefault="unqualified">
  <xsd:annotation>
    <xsd:documentation xml:lang="en">
      Schema for Defining LFB Classes and associated types (frames,
      data types for LFB attributes, and metadata).
    </xsd:documentation>
  </xsd:annotation>
  <xsd:element name="description" type="xsd:string" />
  <xsd:element name="synopsis" type="xsd:string" />
  <!-- Document root element: LFBLibrary -->
  <xsd:element name="LFBLibrary">
    <xsd:complexType>
      <xsd:sequence>
        <xsd:element ref="description" minOccurs="0" />
        <xsd:element name="load" type="loadType"
          minOccurs="0" maxOccurs="unbounded" />
        <xsd:element name="frameDefs" type="frameDefsType"
          minOccurs="0" />
        <xsd:element name="dataTypeDefs" type="dataTypeDefsType"
          minOccurs="0" />
        <xsd:element name="metadataDefs" type="metadataDefsType"
          minOccurs="0" />
        <xsd:element name="LFBClassDefs" type="LFBClassDefsType"
          minOccurs="0" />
      </xsd:sequence>
      <xsd:attribute name="provides" type="xsd:Name"
        use="required" />
    </xsd:complexType>
    <!-- Uniqueness constraints -->
    <xsd:key name="frame">
      <xsd:selector xpath="lfb:frameDefs/lfb:frameDef" />
      <xsd:field xpath="lfb:name" />
    </xsd:key>
    <xsd:key name="dataType">
      <xsd:selector xpath="lfb:dataTypeDefs/lfb:dataTypeDef" />
      <xsd:field xpath="lfb:name" />
    </xsd:key>
    <xsd:key name="metadataDef">
      <xsd:selector xpath="lfb:metadataDefs/lfb:metadataDef" />
      <xsd:field xpath="lfb:name" />
    </xsd:key>
  </xsd:element>
</xsd:schema>
```

```

    </xsd:key>
    <xsd:key name="metadataDefID">
      <xsd:selector xpath="lfb:metadataDefs/lfb:metadataDef" />
      <xsd:field xpath="lfb:metadataID" />
    </xsd:key>
    <xsd:key name="LFBClassDef">
      <xsd:selector xpath="lfb:LFBClassDefs/lfb:LFBClassDef" />
      <xsd:field xpath="lfb:name" />
    </xsd:key>
    <xsd:key name="LFBClassDefID">
      <xsd:selector xpath="lfb:LFBClassDefs/lfb:LFBClassDef" />
      <xsd:field xpath="@LFBClassID" />
    </xsd:key>
  </xsd:element>
  <xsd:complexType name="loadType">
    <xsd:attribute name="library" type="xsd:Name" use="required" />
    <xsd:attribute name="location" type="xsd:anyURI"
      use="optional" />
  </xsd:complexType>
  <xsd:complexType name="frameDefsType">
    <xsd:sequence>
      <xsd:element name="frameDef" maxOccurs="unbounded">
        <xsd:complexType>
          <xsd:sequence>
            <xsd:element name="name" type="xsd:NMTOKEN" />
            <xsd:element ref="synopsis" />
            <xsd:element ref="description"
              minOccurs="0" />
          </xsd:sequence>
        </xsd:complexType>
      </xsd:element>
    </xsd:sequence>
  </xsd:complexType>
  <xsd:complexType name="dataTypeDefsType">
    <xsd:sequence>
      <xsd:element name="dataTypeDef" maxOccurs="unbounded">
        <xsd:complexType>
          <xsd:sequence>
            <xsd:element name="name" type="xsd:NMTOKEN" />
            <xsd:element name="derivedFrom" type="xsd:NMTOKEN"
              minOccurs="0" />
            <xsd:element ref="synopsis" />
            <xsd:element ref="description"
              minOccurs="0" />
            <xsd:group ref="typeDeclarationGroup" />
          </xsd:sequence>
        </xsd:complexType>
      </xsd:element>
    </xsd:sequence>
  </xsd:complexType>

```

```

    </xsd:sequence>
</xsd:complexType>
<!-- Predefined (built-in) atomic data-types are: char, uchar,
      int16, uint16, int32, uint32, int64, uint64, string[N], string,
      byte[N], boolean, octetstring[N], float32, float64 -->
<xsd:group name="typeDeclarationGroup">
  <xsd:choice>
    <!-- Extension -->
    <xsd:sequence>
      <!-- /Extension -->
      <xsd:element name="typeRef" type="typeRefNMTOKEN" />
      <!-- Extension -->
      <xsd:element name="DefaultValue" type="xsd:token"
        minOccurs="0" />
    </xsd:sequence>
    <!-- /Extension -->
    <xsd:element name="atomic" type="atomicType" />
    <xsd:element name="array" type="arrayType">
      <!-- Extension -->
      <!--declare keys to have unique IDs -->
      <xsd:key name="contentKeyID">
        <xsd:selector xpath="lfb:contentKey" />
        <xsd:field xpath="@contentKeyID" />
      </xsd:key>
      <!-- /Extension -->
    </xsd:element>
    <xsd:element name="struct" type="structType">
      <!-- Extension -->
      <!-- key for componentIDs uniqueness in a struct -->
      <xsd:key name="structComponentID">
        <xsd:selector xpath="lfb:component" />
        <xsd:field xpath="@componentID" />
      </xsd:key>
      <!-- /Extension -->
    </xsd:element>
    <xsd:element name="union" type="structType" />
    <xsd:element name="alias" type="typeRefNMTOKEN" />
  </xsd:choice>
</xsd:group>
<xsd:simpleType name="typeRefNMTOKEN">
  <xsd:restriction base="xsd:token">
    <xsd:pattern value="\c+" />
    <xsd:pattern value="string\[\\d+\\]" />
    <xsd:pattern value="byte\[\\d+\\]" />
    <xsd:pattern value="octetstring\[\\d+\\]" />
  </xsd:restriction>
</xsd:simpleType>
<xsd:complexType name="atomicType">

```

```
<xsd:sequence>
  <xsd:element name="baseType" type="typeRefNMTOKEN" />
  <xsd:element name="rangeRestriction"
    type="rangeRestrictionType" minOccurs="0" />
  <xsd:element name="specialValues" type="specialValuesType"
    minOccurs="0">
    <!-- Extension -->
    <xsd:key name="SpecialValue">
      <xsd:selector xpath="specialValue" />
      <xsd:field xpath="@value" />
    </xsd:key>
    <!-- /Extension -->
  </xsd:element>
  <!-- Extension -->
  <xsd:element name="defaultValue" type="xsd:token"
    minOccurs="0" />
  <!-- /Extension -->
</xsd:sequence>
</xsd:complexType>
<xsd:complexType name="rangeRestrictionType">
  <xsd:sequence>
    <xsd:element name="allowedRange" maxOccurs="unbounded">
      <xsd:complexType>
        <xsd:attribute name="min" type="xsd:integer"
          use="required" />
        <xsd:attribute name="max" type="xsd:integer"
          use="required" />
      </xsd:complexType>
    </xsd:element>
  </xsd:sequence>
</xsd:complexType>
<xsd:complexType name="specialValuesType">
  <xsd:sequence>
    <xsd:element name="specialValue" maxOccurs="unbounded">
      <xsd:complexType>
        <xsd:sequence>
          <xsd:element name="name" type="xsd:NMTOKEN" />
          <xsd:element ref="synopsis" />
        </xsd:sequence>
        <xsd:attribute name="value" type="xsd:token" />
      </xsd:complexType>
    </xsd:element>
  </xsd:sequence>
</xsd:complexType>
<xsd:complexType name="arrayType">
  <xsd:sequence>
    <xsd:group ref="typeDeclarationGroup" />
    <xsd:element name="contentKey" minOccurs="0"
```

```

        maxOccurs="unbounded">
        <xsd:complexType>
            <xsd:sequence>
                <xsd:element name="contentKeyField"
                    type="xsd:string" maxOccurs="unbounded" />
            </xsd:sequence>
            <xsd:attribute name="contentKeyID" type="xsd:integer"
                use="required" />
        </xsd:complexType>
    </xsd:element>
</xsd:sequence>
<xsd:attribute name="type" use="optional"
    default="variable-size">
    <xsd:simpleType>
        <xsd:restriction base="xsd:string">
            <xsd:enumeration value="fixed-size" />
            <xsd:enumeration value="variable-size" />
        </xsd:restriction>
    </xsd:simpleType>
</xsd:attribute>
<xsd:attribute name="length" type="xsd:integer"
    use="optional" />
<xsd:attribute name="maxLength" type="xsd:integer"
    use="optional" />
</xsd:complexType>
<xsd:complexType name="structType">
    <xsd:sequence>
        <xsd:element name="derivedFrom" type="typeRefNMTOKEN"
            minOccurs="0" />
        <xsd:element name="component" maxOccurs="unbounded">
            <xsd:complexType>
                <xsd:sequence>
                    <xsd:element name="name" type="xsd:NMTOKEN" />
                    <xsd:element ref="synopsis" />
                    <xsd:element ref="description"
                        minOccurs="0" />
                    <xsd:element name="optional" minOccurs="0" />
                    <xsd:group ref="typeDeclarationGroup" />
                </xsd:sequence>
                <xsd:attribute name="componentID"
                    type="xsd:unsignedInt" use="required" />
            </xsd:complexType>
        </xsd:element>
    </xsd:sequence>
</xsd:complexType>
<xsd:complexType name="metadataDefsType">
    <xsd:sequence>
        <xsd:element name="metadataDef" maxOccurs="unbounded">

```

```

<xsd:complexType>
  <xsd:sequence>
    <xsd:element name="name" type="xsd:NMTOKEN" />
    <xsd:element ref="synopsis" />
    <xsd:element name="metadataID" type="xsd:integer"/>
    <xsd:element ref="description"
      minOccurs="0" />
    <xsd:choice>
      <xsd:element name="typeRef"
        type="typeRefNMTOKEN" />
      <xsd:element name="atomic" type="atomicType" />
      <!-- Extension -->
      <xsd:element name="array" type="arrayType">
        <!--declare keys to have unique IDs -->
        <xsd:key name="contentKeyID1">
          <xsd:selector xpath="lfb:contentKey" />
          <xsd:field xpath="@contentKeyID" />
        </xsd:key>
        <!-- /Extension -->
      </xsd:element>
      <xsd:element name="struct" type="structType">
        <!-- Extension -->
        <!-- key declaration to make componentIDs
          unique in a struct -->
        <xsd:key name="structComponentID1">
          <xsd:selector xpath="lfb:component" />
          <xsd:field xpath="@componentID" />
        </xsd:key>
        <!-- /Extension -->
      </xsd:element>
    </xsd:choice>
  </xsd:sequence>
</xsd:complexType>
</xsd:element>
</xsd:sequence>
</xsd:complexType>
<xsd:complexType name="LFBClassDefsType">
  <xsd:sequence>
    <xsd:element name="LFBClassDef" maxOccurs="unbounded">
      <xsd:complexType>
        <xsd:sequence>
          <xsd:element name="name" type="xsd:NMTOKEN" />
          <xsd:element ref="synopsis" />
          <xsd:element name="version" type="versionType" />
          <xsd:element name="derivedFrom" type="xsd:NMTOKEN"
            minOccurs="0" />
          <xsd:element name="inputPorts"
            type="inputPortsType"

```



```
        minOccurs="0" />
<xsd:element name="outputPorts"
  type="outputPortsType"
  minOccurs="0" />
<xsd:element name="components"
  type="LFBComponentsType"
  minOccurs="0" />
<xsd:element name="capabilities"
  type="LFBCapabilitiesType"
  minOccurs="0" />
<xsd:element name="events" type="eventsType"
  minOccurs="0" />
<xsd:element ref="description"
  minOccurs="0" />
</xsd:sequence>
<xsd:attribute name="LFBClassID" type="xsd:unsignedInt"
  use="required" />
</xsd:complexType>
<!-- Key constraint to ensure unique attribute names
  within a class: -->
<xsd:key name="components">
  <xsd:selector xpath="lfb:components/lfb:component" />
  <xsd:field xpath="lfb:name" />
</xsd:key>
<xsd:key name="capabilities">
  <xsd:selector xpath="lfb:capabilities/lfb:capability"/>
  <xsd:field xpath="lfb:name" />
</xsd:key>
<xsd:key name="events">
  <xsd:selector xpath="lfb:events/lfb:event" />
  <xsd:field xpath="lfb:name" />
</xsd:key>
<xsd:key name="eventsIDs">
  <xsd:selector xpath="lfb:events/lfb:event" />
  <xsd:field xpath="@eventID" />
</xsd:key>
<xsd:key name="componentIDs">
  <xsd:selector xpath="lfb:components/lfb:component" />
  <xsd:field xpath="@componentID" />
</xsd:key>
<xsd:key name="capabilityIDs">
  <xsd:selector xpath="lfb:capabilities/lfb:capability"/>
  <xsd:field xpath="@componentID" />
</xsd:key>
<xsd:key name="ComponentCapabilityComponentIDUniqueness">
  <xsd:selector
    xpath="lfb:components/lfb:component |
    lfb:capabilities/lfb:capability|lfb:events" />
```

```

        <xsd:field xpath="@componentID|@baseID" />
      </xsd:key>
    </xsd:element>
  </xsd:sequence>
</xsd:complexType>
<xsd:simpleType name="versionType">
  <xsd:restriction base="xsd:NMTOKEN">
    <xsd:pattern value="[1-9][0-9]*\.[1-9][0-9]*|0" />
  </xsd:restriction>
</xsd:simpleType>
<xsd:complexType name="inputPortsType">
  <xsd:sequence>
    <xsd:element name="inputPort" type="inputPortType"
      maxOccurs="unbounded" />
  </xsd:sequence>
</xsd:complexType>
<xsd:complexType name="inputPortType">
  <xsd:sequence>
    <xsd:element name="name" type="xsd:NMTOKEN" />
    <xsd:element ref="synopsis" />
    <xsd:element name="expectation" type="portExpectationType"/>
    <xsd:element ref="description" minOccurs="0" />
  </xsd:sequence>
  <xsd:attribute name="group" type="xsd:boolean"
    use="optional" default="0" />
</xsd:complexType>
<xsd:complexType name="portExpectationType">
  <xsd:sequence>
    <xsd:element name="frameExpected" minOccurs="0">
      <xsd:complexType>
        <xsd:sequence>
          <!-- ref must refer to a name of a defined
            frame type -->
          <xsd:element name="ref" type="xsd:string"
            maxOccurs="unbounded" />
        </xsd:sequence>
      </xsd:complexType>
    </xsd:element>
    <xsd:element name="metadataExpected" minOccurs="0">
      <xsd:complexType>
        <xsd:choice maxOccurs="unbounded">
          <!-- ref must refer to a name of a defined
            metadata -->
          <xsd:element name="ref"
            type="metadataInputRefType" />
          <xsd:element name="one-of"
            type="metadataInputChoiceType" />
        </xsd:choice>
      </xsd:complexType>
    </xsd:element>
  </xsd:sequence>
</xsd:complexType>

```

```
        </xsd:complexType>
      </xsd:element>
    </xsd:sequence>
  </xsd:complexType>
  <xsd:complexType name="metadataInputChoiceType">
    <xsd:choice minOccurs="2" maxOccurs="unbounded">
      <!-- ref must refer to a name of a defined metadata -->
      <xsd:element name="ref" type="xsd:NMTOKEN" />
      <xsd:element name="one-of" type="metadataInputChoiceType" />
      <xsd:element name="metadataSet" type="metadataInputSetType"/>
    </xsd:choice>
  </xsd:complexType>
  <xsd:complexType name="metadataInputSetType">
    <xsd:choice minOccurs="2" maxOccurs="unbounded">
      <!-- ref must refer to a name of a defined metadata -->
      <xsd:element name="ref" type="metadataInputRefType" />
      <xsd:element name="one-of" type="metadataInputChoiceType"/>
    </xsd:choice>
  </xsd:complexType>
  <xsd:complexType name="metadataInputRefType">
    <xsd:simpleContent>
      <xsd:extension base="xsd:NMTOKEN">
        <xsd:attribute name="dependency" use="optional"
          default="required">
          <xsd:simpleType>
            <xsd:restriction base="xsd:string">
              <xsd:enumeration value="required" />
              <xsd:enumeration value="optional" />
            </xsd:restriction>
          </xsd:simpleType>
        </xsd:attribute>
        <xsd:attribute name="defaultValue" type="xsd:token"
          use="optional" />
      </xsd:extension>
    </xsd:simpleContent>
  </xsd:complexType>
  <xsd:complexType name="outputPortsType">
    <xsd:sequence>
      <xsd:element name="outputPort" type="outputPortType"
        maxOccurs="unbounded" />
    </xsd:sequence>
  </xsd:complexType>
  <xsd:complexType name="outputPortType">
    <xsd:sequence>
      <xsd:element name="name" type="xsd:NMTOKEN" />
      <xsd:element ref="synopsis" />
      <xsd:element name="product" type="portProductType" />
      <xsd:element ref="description" minOccurs="0" />
    </xsd:sequence>
  </xsd:complexType>

```

```
</xsd:sequence>
  <xsd:attribute name="group" type="xsd:boolean"
    use="optional" default="0" />
</xsd:complexType>
<xsd:complexType name="portProductType">
  <xsd:sequence>
    <xsd:element name="frameProduced" minOccurs="0">
      <xsd:complexType>
        <xsd:sequence>
          <!-- ref must refer to a name of a defined
            frame type -->
          <xsd:element name="ref" type="xsd:NMTOKEN"
            maxOccurs="unbounded" />
        </xsd:sequence>
      </xsd:complexType>
    </xsd:element>
    <xsd:element name="metadataProduced" minOccurs="0">
      <xsd:complexType>
        <xsd:choice maxOccurs="unbounded">
          <!-- ref must refer to a name of a defined
            metadata -->
          <xsd:element name="ref"
            type="metadataOutputRefType" />
          <xsd:element name="one-of"
            type="metadataOutputChoiceType" />
        </xsd:choice>
      </xsd:complexType>
    </xsd:element>
  </xsd:sequence>
</xsd:complexType>
<xsd:complexType name="metadataOutputChoiceType">
  <xsd:choice minOccurs="2" maxOccurs="unbounded">
    <!-- ref must refer to a name of a defined metadata -->
    <xsd:element name="ref" type="xsd:NMTOKEN" />
    <xsd:element name="one-of" type="metadataOutputChoiceType" />
    <xsd:element name="metadataSet"
      type="metadataOutputSetType" />
  </xsd:choice>
</xsd:complexType>
<xsd:complexType name="metadataOutputSetType">
  <xsd:choice minOccurs="2" maxOccurs="unbounded">
    <!-- ref must refer to a name of a defined metadata -->
    <xsd:element name="ref" type="metadataOutputRefType" />
    <xsd:element name="one-of"
      type="metadataOutputChoiceType" />
  </xsd:choice>
</xsd:complexType>
<xsd:complexType name="metadataOutputRefType">
```

```
<xsd:simpleContent>
  <xsd:extension base="xsd:NMTOKEN">
    <xsd:attribute name="availability" use="optional"
      default="unconditional">
      <xsd:simpleType>
        <xsd:restriction base="xsd:string">
          <xsd:enumeration value="unconditional" />
          <xsd:enumeration value="conditional" />
        </xsd:restriction>
      </xsd:simpleType>
    </xsd:attribute>
  </xsd:extension>
</xsd:simpleContent>
</xsd:complexType>
<xsd:complexType name="LFBComponentsType">
  <xsd:sequence>
    <xsd:element name="component" maxOccurs="unbounded">
      <xsd:complexType>
        <xsd:sequence>
          <xsd:element name="name" type="xsd:NMTOKEN" />
          <xsd:element ref="synopsis" />
          <xsd:element ref="description"
            minOccurs="0" />
          <xsd:element name="optional" minOccurs="0" />
          <xsd:group ref="typeDeclarationGroup" />
          <xsd:element name="defaultValue" type="xsd:token"
            minOccurs="0" />
        </xsd:sequence>
        <xsd:attribute name="access" use="optional"
          default="read-write">
          <xsd:simpleType>
            <xsd:list itemType="accessModeType" />
          </xsd:simpleType>
        </xsd:attribute>
        <xsd:attribute name="componentID"
          type="xsd:unsignedInt" use="required" />
      </xsd:complexType>
    </xsd:element>
  </xsd:sequence>
</xsd:complexType>
<xsd:simpleType name="accessModeType">
  <xsd:restriction base="xsd:NMTOKEN">
    <xsd:enumeration value="read-only" />
    <xsd:enumeration value="read-write" />
    <xsd:enumeration value="write-only" />
    <xsd:enumeration value="read-reset" />
    <xsd:enumeration value="trigger-only" />
  </xsd:restriction>
</xsd:simpleType>
```

```
</xsd:simpleType>
<xsd:complexType name="LFBCapabilitiesType">
  <xsd:sequence>
    <xsd:element name="capability" maxOccurs="unbounded">
      <xsd:complexType>
        <xsd:sequence>
          <xsd:element name="name" type="xsd:NMTOKEN" />
          <xsd:element ref="synopsis" />
          <xsd:element ref="description"
            minOccurs="0" />
          <xsd:element name="optional" minOccurs="0" />
          <xsd:group ref="typeDeclarationGroup" />
        </xsd:sequence>
        <xsd:attribute name="componentID" type="xsd:integer"
          use="required" />
      </xsd:complexType>
    </xsd:element>
  </xsd:sequence>
</xsd:complexType>
<xsd:complexType name="eventsType">
  <xsd:sequence>
    <xsd:element name="event" maxOccurs="unbounded">
      <xsd:complexType>
        <xsd:sequence>
          <xsd:element name="name" type="xsd:NMTOKEN" />
          <xsd:element ref="synopsis" />
          <xsd:element name="eventTarget"
            type="eventPathType" />
          <xsd:element ref="eventCondition" />
          <xsd:element name="eventReports"
            type="eventReportsType" minOccurs="0" />
          <xsd:element ref="description"
            minOccurs="0" />
        </xsd:sequence>
        <xsd:attribute name="eventID" type="xsd:integer"
          use="required" />
      </xsd:complexType>
    </xsd:element>
  </xsd:sequence>
  <xsd:attribute name="baseID" type="xsd:integer"
    use="optional" />
</xsd:complexType>
<!-- the substitution group for the event conditions -->
<xsd:element name="eventCondition" abstract="true" />
<xsd:element name="eventCreated"
  substitutionGroup="eventCondition" />
<xsd:element name="eventDeleted"
  substitutionGroup="eventCondition" />
```

```
<xsd:element name="eventChanged"
  substitutionGroup="eventCondition" />
<xsd:element name="eventGreaterThanOr"
  substitutionGroup="eventCondition" />
<xsd:element name="eventLessThan"
  substitutionGroup="eventCondition" />
<xsd:complexType name="eventPathType">
  <xsd:sequence>
    <xsd:element ref="eventPathPart" maxOccurs="unbounded" />
  </xsd:sequence>
</xsd:complexType>
<!-- the substitution group for the event path parts -->
<xsd:element name="eventPathPart" type="xsd:string"
  abstract="true" />
<xsd:element name="eventField" type="xsd:string"
  substitutionGroup="eventPathPart" />
<xsd:element name="eventSubscript" type="xsd:string"
  substitutionGroup="eventPathPart" />
<xsd:complexType name="eventReportsType">
  <xsd:sequence>
    <xsd:element name="eventReport" type="eventPathType"
      maxOccurs="unbounded" />
  </xsd:sequence>
</xsd:complexType>
<xsd:simpleType name="booleanType">
  <xsd:restriction base="xsd:string">
    <xsd:enumeration value="0" />
    <xsd:enumeration value="1" />
  </xsd:restriction>
</xsd:simpleType>
</xsd:schema>
```

OpenFlow XML Library

6. Acknowledgements

TBD

7. IANA Considerations

This memo includes no request to IANA.

8. Security Considerations

TBD

9. References

9.1. Normative References

- [I-D.haleplidis-forces-openflow-lib]
Haleplidis, E., Cherkaoui, O., Hares, S., and W. Wang,
"Forwarding and Control Element Separation (ForCES)
OpenFlow Model Library",
draft-haleplidis-forces-openflow-lib-01 (work in
progress), July 2012.
- [OpenFlowSpec1.1]
<http://www.OpenFlow.org/>, "The OpenFlow 1.1
Specification.", <[http://www.OpenFlow.org/documents/
OpenFlow-spec-v1.1.0.pdf](http://www.OpenFlow.org/documents/OpenFlow-spec-v1.1.0.pdf)>.
- [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.

Author's Address

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

Email: ehalep@ece.upatras.gr

Internet Engineering Task Force
Internet-Draft
Intended status: Informational
Expires: April 22, 2013

E. Haleplidis
University of Patras
J. Halpern
Ericsson
October 19, 2012

ForCES Packet Parallelization
draft-haleplidis-forces-packet-parallelization-01

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, so that control elements (CEs) can control the FEs accordingly. More specifically, the model describes the logical functions that are present in an FE, what capabilities these functions support, and how these functions are or can be interconnected.

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 22, 2013.

Copyright Notice

Copyright (c) 2012 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	5
3. Packet Parallelization	6
4. Parallel Base Types	12
4.1. Frame Types	12
4.2. Data Types	12
4.3. MetaData Types	12
5. Parallel LFBs	14
5.1. Splitter	14
5.1.1. Data Handling	14
5.1.2. Components	14
5.1.3. Capabilities	15
5.1.4. Events	15
5.2. Merger	15
5.2.1. Data Handling	15
5.2.2. Components	16
5.2.3. Capabilities	16
5.2.4. Events	16
6. XML for Parallel LFB library	17
7. Acknowledgements	24
8. IANA Considerations	25
9. Security Considerations	26
10. References	27
10.1. Normative References	27
10.2. Informative References	27
Authors' Addresses	28

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.

Element - Element is generally used in this document in accordance with the XML usage of the term. It refers to an XML tagged part of an XML document. For a precise definition, please see the full set of XML specifications from the W3C. This term is included in

this list for completeness because the ForCES formal model uses XML.

Attribute - Attribute is used in the ForCES formal modeling in accordance with standard XML usage of the term, i.e., to provide attribute information included in an XML tag.

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 Class Library - The LFB class library is a set of LFB classes that has been identified as the most common functions found in most FEs and hence should be defined first by the ForCES Working Group.

2. Introduction

A lot of network devices can process packets in a parallel manner. The 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 modelling concept has been influenced by Cilc. Cilc 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 Cilc concepts used in this document is spawn and sync. Spawn being the place where parallel work can start and sync being the place where the parallel work finishes and must collect all parallel output.

3. 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.

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

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

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 an 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. This modelling framework however allows for more complex parallel datapath topologies as can be seen in Figure 3 which shows one of the parallel paths to be further splitted into a new parallel section.

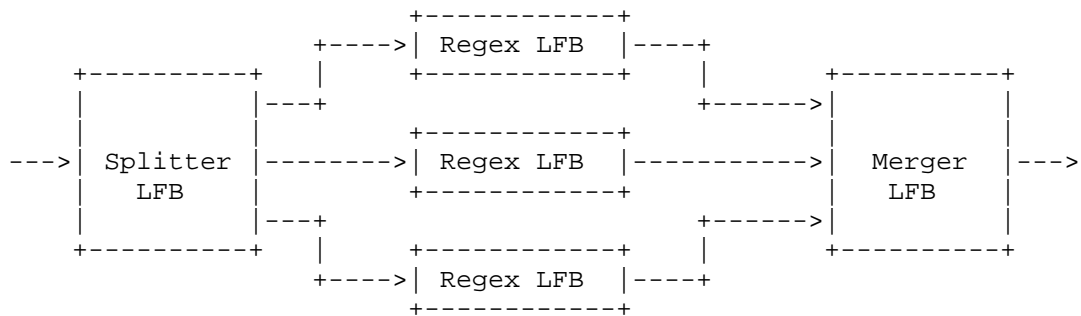


Figure 1: Simple split parallel processing

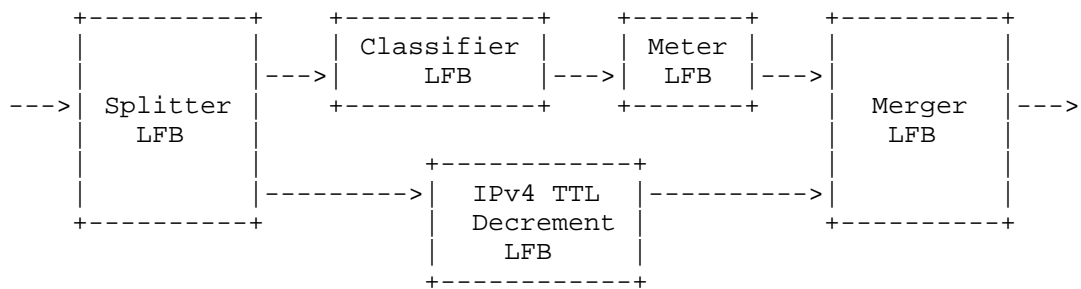


Figure 2: Simple flood parallel processing

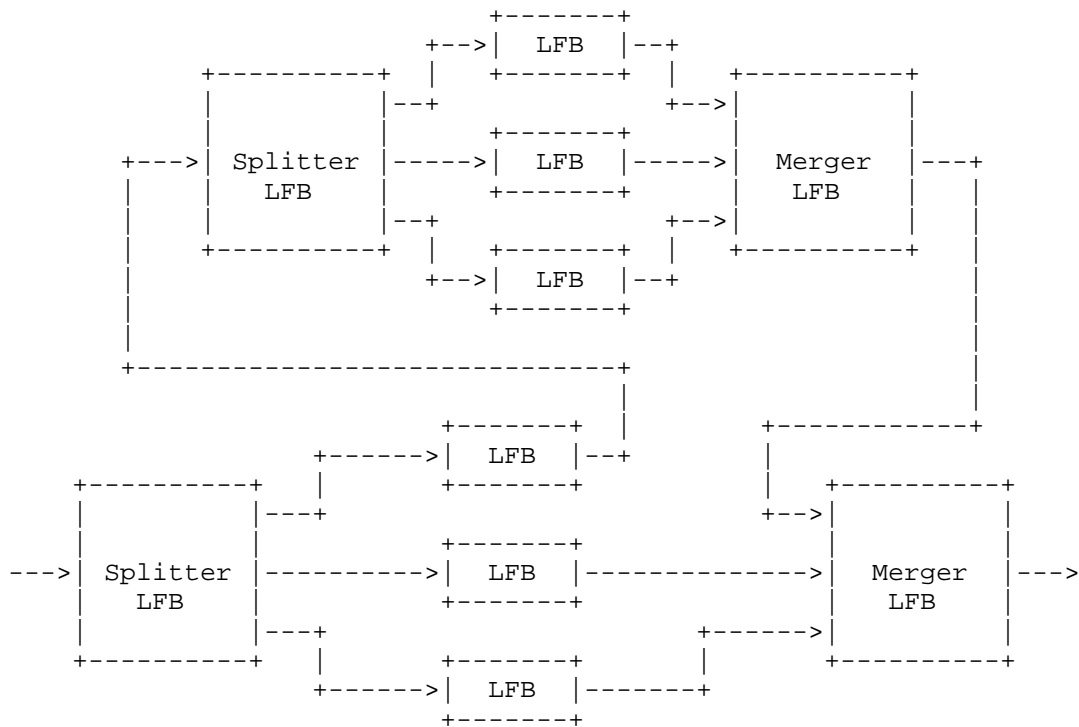


Figure 3: Complex parallel processing

One important element to a developer is the ability to define which LFBs can be used in a parallel mode, with which other LFBs can they be parallelized with and the order of the LFBs can be assembled. This information must be accessible in the core LFBs and therefore this document needs to append one more capability in the FEObject LFB. The topology of the parallel datapath can be deferred and manipulated from the FEObject LFB's LFBTopology.

The FEObject LFB currently specifies the LFBTopology and supported LFBs in an FE. In order to support parallelization the following component is needed 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>LFBParallelOccurenceLimit</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 this parallel LFB
        class can follow in a parallel pipeline</synopsis>
      <optional />
      <array>
        <typeRef>uint32</typeRef>
      </array>
    </component>
  </struct>
</dataTypeDef>
```

```

    </component>
    <component componentID="6">
      <name>AllowedParallelBefore</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 4: XML Definition for FEObjectLFB extension

While the ForCES model cannot describe how the splitting or the merging is actually done as that is an implementation issue of the actual LFB, however this document defines operational parameters to control 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. Additionally this document defines metadata, which contain necessary information to assist the merging procedure. The following metadata are defined:

1. ParallelType - Flood or split
2. Correlator - Identify packets or chunks that belonged to the initial packet that entered the Splitter LFB

3. ParallelNum - Number of packet or chunk for specific Correlator.
4. ParallelPartsCount - Total number of packets or chunks for specific Correlator.

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 case that in a parallel path there is an additional Splitter LFB therefore parallelizing even more that path, a new set of metadata MUST be produced for that specific Splitter and the first set of metadata MUST be tunneled through without being consumed or changed until reaching the corresponding Merger LFB where it will be sent out again in the previous parallel path.

In case of a packet/chunk being branded invalid by an LFB in a parallel path, it MUST be sent by an output port of said LFB

An LFB inside a parallel path decides that a packet or a chunk has to be dropped it MAY drop it 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.

4. Parallel Base Types

4.1. 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

4.2. 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

4.3. MetaData Types

The following metadata are defined in the OpenFlow type library:

Metadata Name	Type	ID	Synopsis
ParallelType	uchar	32	The type of parallelization this packet will go through. 0 for flood, 1 for split.
Correlator	uint32	33	An identification number to specify that packets or chunks belong to the same parallel work.

ParallelNum	uint32	34	Defines the number of the specific packet or chunk of the specific parallel ID.
ParallelPartsCount	uint32	35	Defines the total number of packets or chunks for the specific parallel ID.

Metadata Structure for Merging

5. Parallel LFBs

5.1. Splitter

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

5.1.1. 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 ParallelOut'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. All chunks will be sent out in a round-robin fashion through the group output ParallelOut's instances. Each packet or chunk will be accompanied by the following metadata:

- o ParallelType - The paralleltype split or flood.
- o Parallel ID - generated by the splitter LFB to identify that chunks or packets belong to the same parallel work.
- o Parallel Num - 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 parallel id.
- o Valid - with a default value of true. The merger LFB must know if a packet or a chunk must be set invalid by an LFB in one part of the parallel pipeline.

5.1.2. Components

This LFB has only two components specified. 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 maximum size of a chunk when a packet is split into multiple same size chunks.

5.1.3. Capabilities

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

5.1.4. Events

This LFB has no events specified.

5.2. Merger

A merger LFB receives multiple packets or multiple chunks of the same packet and merge them into one merged packet.

5.2.1. Data Handling

The Merger LFB receives either a packet or a chunk via the group input ParallelIn, along with the ParallelType metadata to identify whether what was received was a packet or a chunk, the Correlator, the ParallelNum and the ParallelPartsCount.

In case that an LFB has dropped a packet or a chunk within a parallel path 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. Current defined metadata's in the Base LFB Library [I-D.ietf-forces-lfb-lib] are the ExceptionID and the ValidateErrorID. The Merger LFB MAY store the parallel metadata along with the exception metadata as a string in the optional InvalideMetadataSets as a means for the CE to debug errors in the parallel path.

If the MergeWaitType is set to false the Merger LFB will initiate the merge process upon receiving the first packet. If false it will wait for all packet in the Correlator to arrive.

If one packet or chunk has been received through the InvalidIn port then the merging procedure will be operate as configured by the InvalidAction component. 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 finished the resulting packet will be sent via the singleton output port PacketOutput.

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

5.2.2. 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. `MergeWaitType` - a boolean. If true the Merger LFB will wait for all packets or chunks to be received prior to sending out a response. If false, when one packet or a chunk with a response is received by the merge LFB it will start with the merge process.
3. `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.
4. `InvalidAllCounter` - a uint 32 that counts the number of merges where all packets/chunks entered the merger LFB through the `InvalidIn` input port.
5. `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`.
6. `InvalidMetadataSets` - an array of strings. An optional component that stores metadata sets along with the error id as a string. This could provide a debug information to the CE regarding errors in the parallel paths.

5.2.3. Capabilities

This LFB has no capabilities specified.

5.2.4. 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.

6. XML for Parallel LFB library

```

<?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"
  xsi:schemaLocation="urn:ietf:params:xml:ns:forces:lfbmodel:1.0"
  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>
  </dataTypeDefs>
  <metadataDefs>
    <metadataDef>
      <name>ParallelType</name>
      <synopsis>The type of parallelization this packet/chunk has
        gone through</synopsis>
      <metadataID>32</metadataID>
      <typeRef>ParallelTypes</typeRef>
    </metadataDef>
    <metadataDef>
      <name>Correlator</name>

```

```

    <synopsis>An identification number to specify that packets
      or chunks originate from the same packet.</synopsis>
    <metadataID>33</metadataID>
    <typeRef>uint32</typeRef>
  </metadataDef>
  <metadataDef>
    <name>ParallelNum</name>
    <synopsis>Defines the number of the specific packet or chunk
      of the specific parallel ID.</synopsis>
    <metadataID>34</metadataID>
    <typeRef>uint32</typeRef>
  </metadataDef>
  <metadataDef>
    <name>ParallelPartsCount</name>
    <synopsis>Defines the total number of packets or chunks for
      the specific parallel ID.</synopsis>
    <metadataID>35</metadataID>
    <typeRef>uint32</typeRef>
  </metadataDef>
</metadataDefs>
<LFBClassDefs>
  <LFBClassDef LFBClassID="30">
    <name>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>Input</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>ParallelOut</name>
        <synopsis>An 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>
          </frameProduced>
        </product>
      </outputPort>
    </outputPorts>
  </LFBClassDef>
</LFBClassDefs>

```

```

        <ref>Chunk</ref>
    </frameProduced>
    <metadataProduced>
        <ref>ParallelType</ref>
        <ref>Correlator</ref>
        <ref>ParallelNum</ref>
        <ref>ParallelPartsCount</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 splitted 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="31">
    <name>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>ParallelIn</name>
    <synopsis>An parallel input port that accepts packets
      or chunks from all output instances</synopsis>
    <expectation>
      <frameExpected>
        <ref>Arbitrary</ref>
        <ref>Chunk</ref>
      </frameExpected>
      <metadataExpected>
        <ref>ParallelType</ref>
        <ref>Correlator</ref>
        <ref>ParallelNum</ref>
        <ref>ParallelPartsCount</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>PacketOutput</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-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="4" access="read-reset">
    <name>InvalidAllCounter</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="5" 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>
<component componentID="6" access="read-reset">
  <name>InvalidMetadataSets</name>
  <synopsis>Buffers metadata sets along with the error
  id as a string.</synopsis>
  <optional />
  <array>
    <typeRef>string</typeRef>
  </array>
</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>50</eventGreaterThan>
    <eventReports>
      <eventReport>
        <eventField>InvalidMergesCounter</eventField>
      </eventReport>
    </eventReports>
  </event>
  <event eventID="2">
    <name>ManyAllInvalids</name>
    <synopsis>An event that specifies if there are too
```

```
        many invalids</synopsis>
    <eventTarget>
        <eventField>InvalidCounter</eventField>
    </eventTarget>
    <eventGreaterThan>50</eventGreaterThan>
    <eventReports>
        <eventReport>
            <eventField>InvalidAllCounter</eventField>
        </eventReport>
    </eventReports>
</event>
</events>
</LFBClassDef>
</LFBClassDefs>
</LFBLibrary>
```

Figure 5: Parallel LFB library

7. Acknowledgements

The authors would like to thank Jamal Hadi Salim for discussions that made this document better.

8. IANA Considerations

This memo includes no request to IANA.

9. Security Considerations

10. References

10.1. Normative References

- [I-D.haleplidis-forces-openflow-lib]
Haleplidis, E., Cherkaoui, O., Hares, S., and W. Wang,
"Forwarding and Control Element Separation (ForCES)
OpenFlow Model Library",
draft-haleplidis-forces-openflow-lib-01 (work in
progress), July 2012.
- [I-D.ietf-forces-lfb-lib]
Wang, W., Haleplidis, E., Ogawa, K., Li, C., and J.
Halpern, "ForCES Logical Function Block (LFB) Library",
draft-ietf-forces-lfb-lib-08 (work in progress),
February 2012.
- [OpenFlowSpec1.1]
<http://www.OpenFlow.org/>, "The OpenFlow 1.1
Specification.", <[http://www.OpenFlow.org/documents/
OpenFlow-spec-v1.1.0.pdf](http://www.OpenFlow.org/documents/OpenFlow-spec-v1.1.0.pdf)>.
- [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.

10.2. Informative References

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

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, 20178
VA

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

Internet Engineering Task Force
Internet-Draft
Intended status: Informational
Expires: August 29, 2013

D. Joachimpillai
Verizon
J. Hadi Salim
Mojatatu Networks
February 25, 2013

ForCES Inter-FE LFB
draft-joachimpillai-forces-interfelfb-01

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, so that control elements (CEs) can control the FEs accordingly. More specifically, the model describes the logical functions that are present in an FE, what capabilities these functions support, and how these functions are or can be interconnected.

At the moment the ForCES charter restricts the LFB topology to be within an FE. This documents 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 August 29, 2013.

Copyright Notice

Copyright (c) 2013 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	5
3.1. Distributing The LFB Topology	7
4. Proposal Overview	8
4.1. Inserting The Inter-FE LFB	8
4.2. Inter-FE connectivity	10
4.2.1. Inter-FE Ethernet connectivity	12
4.2.1.1. Inter-FE Ethernet Connectivity Issues	13
5. Detailed Description of the inter-FE LFB	14
5.1. Data Handling	15
5.2. Metadata	16
5.3. Components	16
5.4. Capabilities	16
5.5. Events	16
5.6. Inter-FE LFB XML	17
6. Acknowledgements	17
7. IANA Considerations	17
8. Security Considerations	17
9. References	17
9.1. Normative References	17
9.2. Informative References	17
Authors' Addresses	17

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. The array (SupportedLFBs) contains information about each supported LFB class that the FE supports. In addition to indicating that the FE supports an LFB class, FEs with modifiable LFB topologies include information about how LFBs of a specified class may be connected to other LFBs. 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 ForCES charter restricts the LFB links in a topology to be within a single FE (intra-FE connectivity) and as such both the relevant capabilities and component definitions in the FEObject LFB are restricted to that scope. This document describes extending the LFB topology across FEs i.e inter-FE connectivity without needing any changes to the ForCES definitions.

3. Problem Scope

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.

The illustration is meant only as an exercise to showcase how data and metadata is sent down or upstream on a graph of LFBs. For this reason, it abstracts out any ports in both directions and talks about a generic ingress and egress LFB. 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[XXX: ref here].

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.

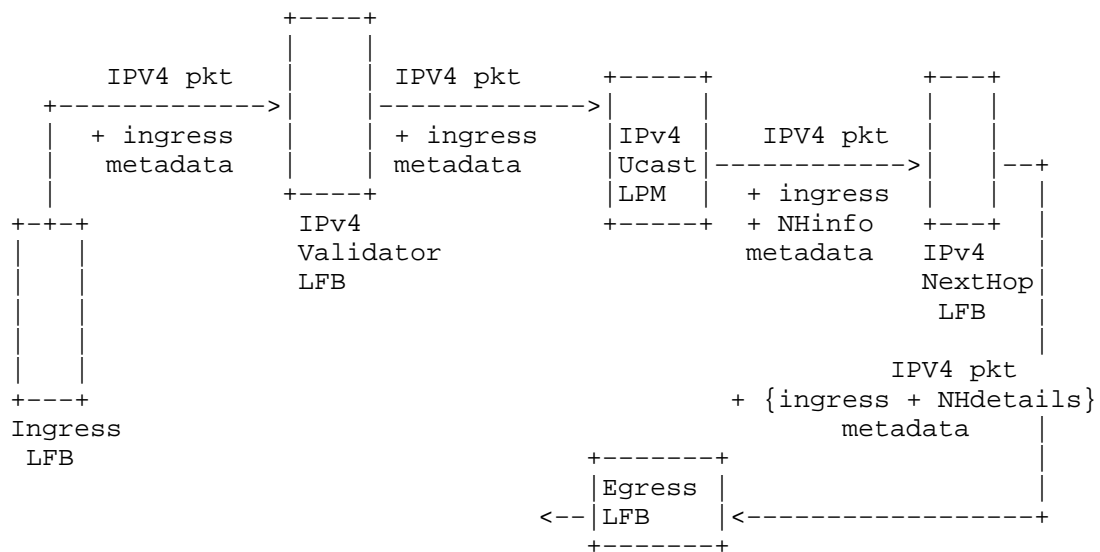


Figure 1: Basic IPV4 packet service LFB topology

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. Distributing The LFB Topology

Figure 2 demonstrates one way the 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.

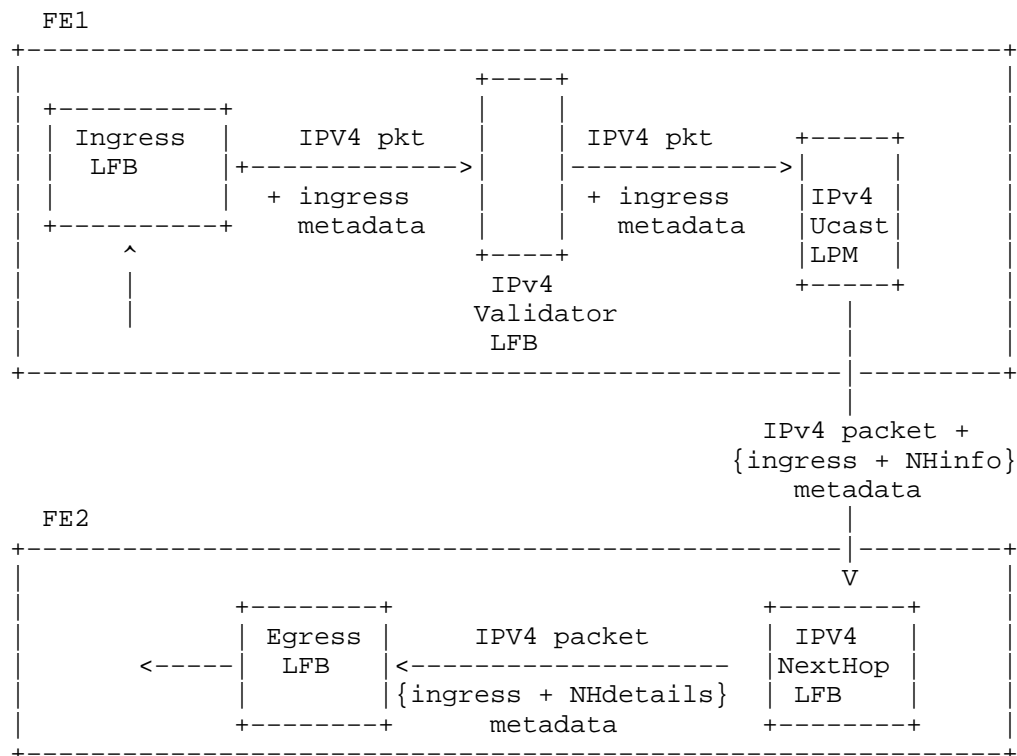


Figure 2: Split IPV4 packet service LFB topology

Some proprietary inter-connect (example Broadcom Higig over XAUI (XXX: ref needed)) maybe used to carry both the IPV4 packet and the related metadata between the IPV4 Unicast LFB and IPV4 NextHop LFB across the two 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 3 to show the topology location where the inter-FE LFB would fit in.

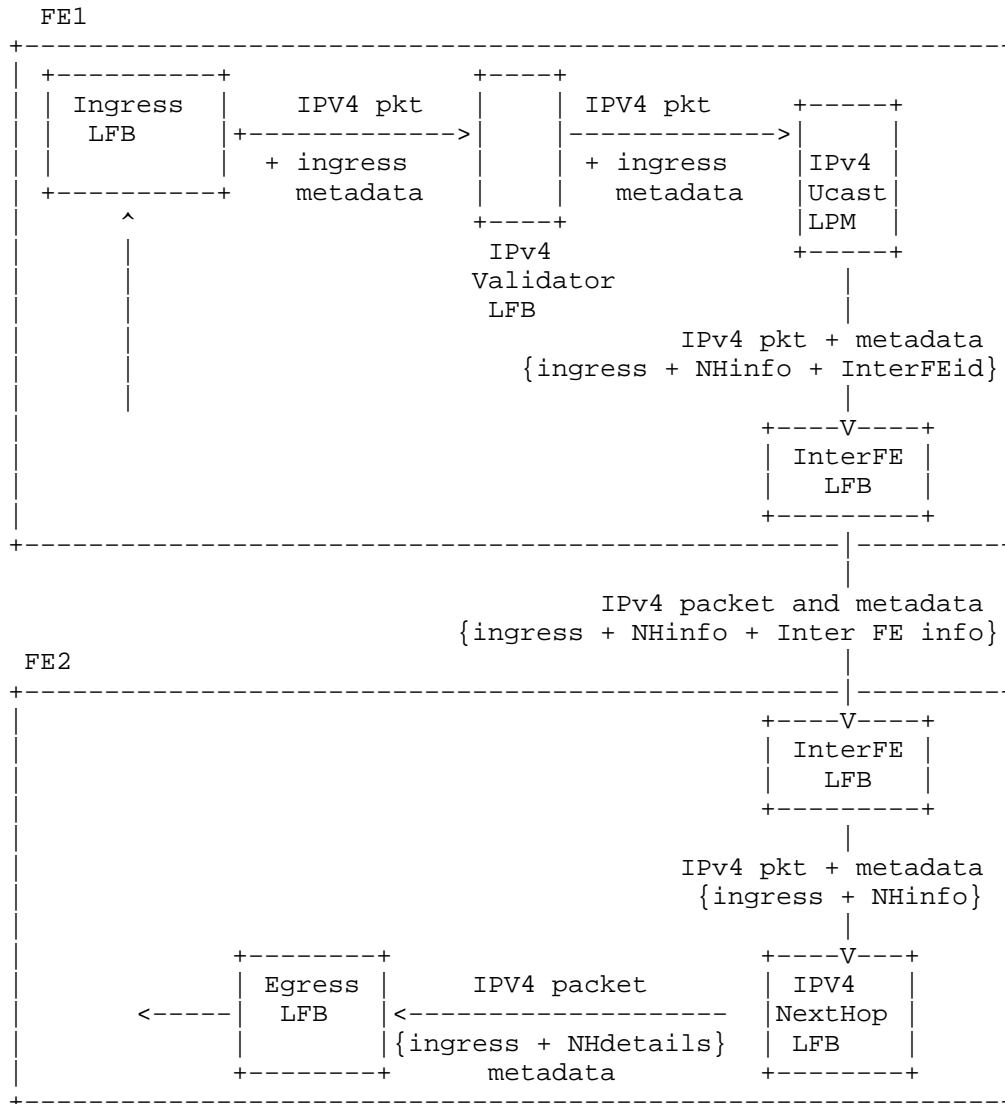


Figure 3: Split IPV4 forwarding service with Inter-FE LFB

As can be observed in Figure 3, 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 and any exception IDs may be passed along with the 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 plus 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 4) 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.

```

+-- T = NESelector-TLV
|
|   +---- NEID
|   |
|   +---- Destination FEID
|   |
|   +---- Source FEID
|
+-- 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 4: Packet format suggestion

XXX: We are going to need two new ForCES TLVs to be defined.

The NESelector carries inter-FE information described earlier. In some cases, the NESelector may be left out in the encapsulation activity (by the inter-FE LFB implementation) if it is already implicitly defined or mapping in the transport (eg VLAN/VXLAN or where in the case of look-aside interfaces or proprietary hard-coded connections such as the one shown in Figure 2).

- o The NESelector carries a 32-bit NEID which defaults to 0. It also carries the destination and source FEIDs. This TLV is new to ForCES and sits in the global ForCES TLV namespace.
- 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.

The METADATA and REDIRECTDATA TLV encapsulations are taken directly from [RFC5810] section 7.9.

4.2.1. Inter-FE Ethernet connectivity

It is expected that a variety of transport encapsulations would be applicable to carry the format described in Figure 1. In the case of existing interconnects, a description of a mapping to interpret the inter-FE details and translate into proprietary or legacy formatting would need to be defined. As an example, already a variety of metadata passing encapsulations exist which are proprietary or semi-standard by virtue of being widely deployed. These include the NPF LA-1 (XXX: ref here), Broadcom Higi2 (XXX: ref here), as well as interlaken (XXX: ref here). For any mapping towards these definitions a different document to describe the mapping, one per transport, is expected to be defined.

In this specific document, we describe a format that is to be used over Ethernet. An ethernet type (To be defined) will be used to imply that a wire format is carrying an inter-FE LFB packet.

XXX: The finer details on what the source and destination MAC address selection are left out for the next draft release. Also left out are any load balancing/multi-pathing activities across selections of destinations FEs.

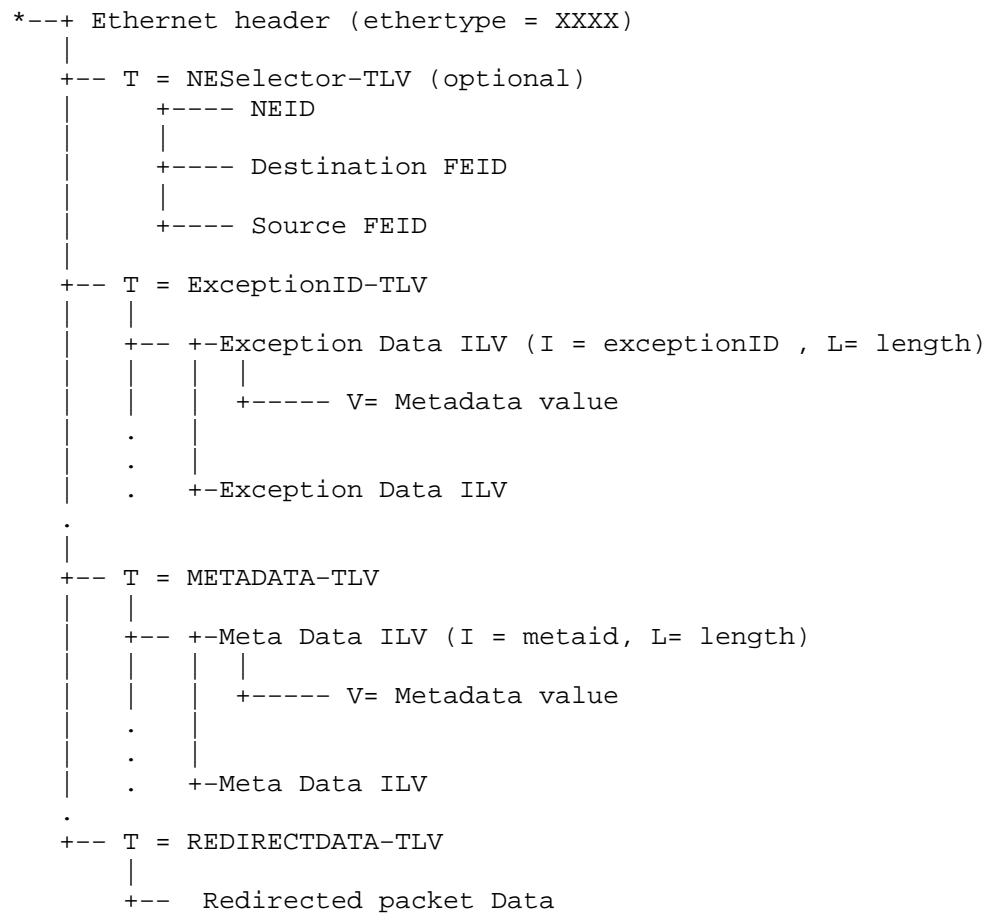


Figure 5: Packet format suggestion

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. There are several possible solutions:
 - * One possible solution is to use large MTUs; however, even that will have limits since the 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. XXX: If we go this path, we would likely need a top level TLV definition to describe the count.
 - * 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.

XXX: These issues will be addressed further in the next draft release. Suggestions welcome.

5. Detailed Description of the inter-FE LFB

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

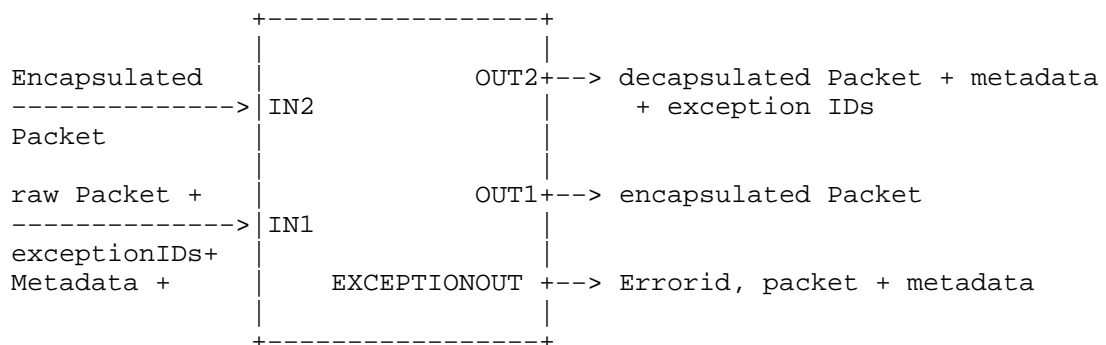


Figure 6: Inter-FE LFB

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 port OUT1 to a downstream LFB or EXCEPTIONOUT 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.

The Inter-FE LFB uses the InterFEid metadatum when on an egress of an FE to lookup the NextFE table. The output result constitutes a matched table row which has the InterFEinfo details i.e. the tuple {NEID, Destination FEID, Source FEID} as well as a filter list which defines 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 and default behavior will be utilized.

In the egress processing case of successful lookup, the inter-FE LFB will:

- o add the NESelector TLV data from the lookup result
- o walk the passed metadatum and encapsulate them within the METADATA-TLV all allowed metadatum.
- o walk all the passed exceptionIDs and encapsulate all allowed exception IDs 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.

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

- o Not add an NESelector TLV
- o walk all the passed metadatum and encapsulate into the METADATA-TLV all metadatum.

- o walk all the passed exceptionIDs and encapsulate all 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.

In the case of ingress processing, the LFB receives an encapsulated packet and extracts the packet data, metadata, and exception IDs.

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 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. Capabilities

XXX: If we support multiple encapsulation methods (other than ethernet), then we could use capabilities to advertise them as different possibilities. It is envisioned then that the NextFE table row will have column indicating to the inter-FE LFB how to encapsulate the different matches. Alternatively this could be left up to the LFB connected in the output port.

5.5. Events

TBA

5.6. Inter-FE LFB XML

TBA

6. Acknowledgements

TBA

7. IANA Considerations

This memo includes no request to IANA.

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

