

Internet Engineering Task Force  
Internet-Draft  
Intended status: Informational  
Expires: January 09, 2014

E. Haleplidis  
University of Patras  
July 08, 2013

ForCES Model Extension  
draft-haleplidis-forces-model-extension-04

Abstract

Forwarding and Control Element Separation (ForCES) defines an architectural framework and associated protocols to standardize information exchange between the control plane and the forwarding plane in a ForCES Network Element (ForCES NE). RFC5812 has defined the ForCES Model 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 small extensions without a need to alter the protocol while retaining backward compatibility with older xml libraries. This document extends the model to allow complex datatypes for metadata, optional default values for datatypes and optional access types for structures. The document also introduces three new features, bitmap as a new datatype, a new event condition BecomesEqualTo and LFB properties.

Status of This Memo

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

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

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

This Internet-Draft will expire on January 09, 2014.

## 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 . . . . .	2
1.1. Requirements Language . . . . .	2
1.2. Definitions . . . . .	2
2. Introduction . . . . .	4
3. ForCES Model Extension proposal . . . . .	4
3.1. Complex datatypes for Metadata . . . . .	4
3.2. Optional Default Value for Datatypes . . . . .	6
3.3. Optional Access Type for Structs . . . . .	7
3.4. New datatype: Bitmap . . . . .	8
3.5. New Event Condition: BecomesEqualTo . . . . .	10
3.6. LFB Properties . . . . .	10
3.7. Enhancing XML Validation . . . . .	11
4. XML Extension Schema for LFB Class Library Documents . . . . .	12
5. Acknowledgements . . . . .	25
6. IANA Considerations . . . . .	25
7. Security Considerations . . . . .	25
8. References . . . . .	25
8.1. Normative References . . . . .	25
8.2. Informative References . . . . .	26
Author's Address . . . . .	26

## 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 more than two years and current experience in its use has demonstrated need for adding new and changing existing modeling concepts.

Specifically this document extends the ForCES Model to allow complex datatypes for metadata, optional default values for datatypes and optional access types for structures. Additionally the document introduces three new features, bitmap as a new datatype, a new event condition BecomesEqualTo and LFB properties.

These extensions are an addendum to the ForCES model [RFC5812] and 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.

XXX: Discussion is needed to specify whether bitmap required protocol definition of how bitmap is sent through the wire.

## 3. ForCES Model Extension proposal

### 3.1. Complex datatypes for 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.

However there are cases where complex metadata are used in the datapath, for example two simple use cases can be seen in the OpenFlow switch 1.1 [OpenFlowSpec1.1] and beyond:

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.

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 Definition 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 Defintion for the schema

### 3.2. Optional Default Value for Datatypes

In the original schema, default values can only be defined for datatypes defined inside LFB components and not inside structures or arrays. Therefore default values of datatypes that are constantly being reused, e.g. counters with default value of 0, have to be constantly respecified. Additionally, datatypes inside complex datatypes cannot be defined with a default value, e.g. a counter inside a struct that has a default value of 0.

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 Defintion 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

### 3.3. Optional Access Type for Structs

In the original schema, the access type can be only be defined on components of LFB and not on components in structs or arrays. However when it's a struct datatype it is not possible to fine-tune access type per component in the struct. A simple use case would be to have a read-write struct component where one of the components is a counter where the access-type could be read-reset or read-only, e.g. a read-reset or a read-only counter inside a struct.

With this extension is it allowed to define the access type for a struct component either in the datatype definitions or in the LFB component definitions.

When the optional access type for a struct component is defined it MUST override the access type of the struct. If by accident an access type for a component in a capability is defined, the access type MUST NOT be taken into account and MUST always be considered as read-only.

This extension alters the definition of the struct in the xml schema from Figure 6 to Figure 7.

```

<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>

```

Figure 6: Initial xml for the struct definition in the schema

```

<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="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>

```

Figure 7: New xml for the struct definition in the schema

### 3.4. New datatype: Bitmap

With the current schema it is valid to create a struct of booleans in order to simulate a bitmap value. However each boolean is sent as 4bytes. This extension adds the bitmap, a set of sequential named bits.



Bitmaps may be useful in describing capabilities, e.g. Link speed capabilities as multiple boolean values.

XXX Discussion may be required as to whether there is a need for protocol description of how the bitmap is sent through the wire.

In the new schema, bits are named followed an optional bit value. An example:

```
<dataTypeDef>
  <name>Bitmap example</name>
  <synopsis>A bitmap field example</synopsis>
  <bitmap>
    <bit name="Bit0" defaultValue="0"/>
    <bit name="Bit1"/>
  </bitmap>
</dataTypeDef>
```

Figure 8: Example of bitmap Defintion

The ordering of the bits MUST be implemented in the order that are defined in the xml library.

The bitmap is defined in the model extension schema is as follows:

```
<xsd:complexType name="bitmapType">
  <xsd:sequence>
    <xsd:element name="bit" maxOccurs="unbounded">
      <xsd:complexType>
        <xsd:attribute name="name" type="xsd:NMTOKEN" use="required"/>
        <xsd:attribute name="defaultValue" type="booleanValues"
          use="optional"></xsd:attribute>
      </xsd:complexType>
    </xsd:element>
  </xsd:sequence>
</xsd:complexType>
<xsd:simpleType name="booleanValues">
  <xsd:restriction base="xsd:integer">
    <xsd:minInclusive value="0"></xsd:minInclusive>
    <xsd:maxInclusive value="1"></xsd:maxInclusive>
  </xsd:restriction>
</xsd:simpleType>
```

Figure 9: New Excerpt of bitmap Defintion in the schema

Along with the needed addition to the typeDeclarationGroup Definition:

```
<xsd:element name="bitmap" type="bitmapType"/>
```

Figure 10: New Excerpt of typeDeclarationGroup Definition in the schema

### 3.5. New Event Condition: BecomesEqualTo

This extension adds one more event condition in the model schema, that of BecomesEqualTo. The difference between Greater Than and Less Than, is that when the value is exactly that of the BecomesEqualTo, the event is triggered. This event condition is particularly useful when there is a need to monitor one or more states of an LFB or the FE. For example in the CEHA [I-D.ietf-forces-ceha] document it may be useful for the master CE to know which backup CEs have just become associated in order to connect to them and begin synchronizing the state of the FE. The master CE could always poll for such information but getting such an event will speed up the process and the event may be useful in other cases as well for monitoring state.

The event MUST be triggered only when the value of the targeted component becomes equal to the event condition value and MUST NOT generate events while the targeted component's value remains equal to the event condition's value.

The BecomesEqualTo is appended to the schema as follows:

```
<xsd:element name="eventBecomesEqualTo"
substitutionGroup="eventCondition"/>
```

Figure 11: New Excerpt of BecomesEqualTo event condition definition in the schema

### 3.6. LFB Properties

The current model definition specifies properties for components of LFBs. Experience however has proven valuable at least for debug reasons, to have statistics per LFB instance to monitor sent/received messages and errors for communication between CE and FE. These properties are read-only.

XXX: Discussion for addressing LFB properties. Possibly in the protocol extension?

The following datatype definitions are to be used as properties for LFB instances.

```
<datatypeDef>
  <name>LFBProperties</name>
```

```
<synopsis>LFB Properties definition</synopsis>
<struct>
  <component componentID="1">
    <name>SentToCE</name>
    <synopsis>Messages sent to CE</synopsis>
    <typeRef>uint32</typeRef>
  </component>
  <component componentID="2">
    <name>SentErrorsToCE</name>
    <synopsis>Error messages sent to CE</synopsis>
    <typeRef>uint32</typeRef>
  </component>
  <component componentID="3">
    <name>ReceivedFromCE</name>
    <synopsis>Messages received from CE</synopsis>
    <typeRef>uint32</typeRef>
  </component>
  <component componentID="4">
    <name>ReceivedErrorsFromCE</name>
    <synopsis>Error messages received from CE</synopsis>
    <typeRef>uint32</typeRef>
  </component>
</struct>
</dataTypeDef>
```

#### Properties for LFB instances

### 3.7. 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.

#### 4. 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: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:element>
</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>
        <xsd:element name="bitmap" type="bitmapType"/>
        <!-- /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: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>
  <!-- Extension -->

```

```
<xsd:complexType name="bitmapType">
  <xsd:sequence>
    <xsd:element name="bit" maxOccurs="unbounded">
      <xsd:complexType>
        <xsd:attribute name="name" type="xsd:NMTOKEN"
          use="required"/>
        <xsd:attribute name="defaultValue" type="booleanValues"
          use="optional"/>
      </xsd:complexType>
    </xsd:element>
  </xsd:sequence>
</xsd:complexType>
<xsd:simpleType name="booleanValues">
  <xsd:restriction base="xsd:integer">
    <xsd:minInclusive value="0"/></xsd:minInclusive>
    <xsd:maxInclusive value="1"/></xsd:maxInclusive>
  </xsd:restriction>
</xsd:simpleType>
<!-- /Extension -->
<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>
        <!-- Extension -->
        <xsd:attribute name="access" use="optional"
          default="read-write">
          <xsd:simpleType>
            <xsd:list itemType="accessModeType" />
          </xsd:simpleType>
        </xsd:attribute>
        <!-- /Extension -->
        <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>
          </xsd:element>
        </xsd:sequence>
      </xsd:complexType>
    </xsd:element>
  </xsd:sequence>
</xsd:complexType>
```

```

        </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>
        </xsd:element>
    </xsd:sequence>
</xsd:complexType>

```

```
</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]*\.[0-9]*" />
  </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 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: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:sequence>
</xsd:complexType>

```

```
<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: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="eventGreaterThan"
    substitutionGroup="eventCondition" />
<xsd:element name="eventLessThan"
    substitutionGroup="eventCondition" />
<!-- Extension -->
    <xsd:element name="eventBecomesEqualTo"
        substitutionGroup="eventCondition"/>
<!-- /Extension -->
<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

## 5. Acknowledgements

The author would like to acknowledge Joel Halpern, Jamal Hadi and Dave Hood for their comments and discussion that helped shape this document in a better way.

## 6. IANA Considerations

This memo includes no request to IANA.

## 7. Security Considerations

The security considerations that have been described in the ForCES Model RFC [RFC5812] apply to this document as well.

## 8. References

### 8.1. Normative References

[I-D.ietf-forces-ceha]  
Ogawa, K., Wang, W., Haleplidis, E., and J. Salim, "ForCES Intra-NE High Availability", draft-ietf-forces-ceha-07 (work in progress), May 2013.

[OpenFlowSpec1.1]

<http://www.OpenFlow.org/>, "The OpenFlow 1.1 Specification.", , <<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.

## 8.2. 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](mailto:ehalep@ece.upatras.gr)

Internet Engineering Task Force  
Internet-Draft  
Intended status: Informational  
Expires: January 06, 2014

J. Hadi Salim  
Mojatatu Networks  
July 05, 2013

ForCES Protocol Extensions  
draft-jhs-forces-PROTOEXTENSION-01

Abstract

Experience in implementing and deploying ForCES architecture has demonstrated need for a few small extensions both to ease programmability and to improve wire efficiency of some transactions. This document describes a few extensions to the ForCES Protocol Specification [RFC5810] semantics to achieve that end goal.

Status of This Memo

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

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

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

This Internet-Draft will expire on January 06, 2014.

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 . . . . .	2
1.1. Requirements Language . . . . .	2
1.2. Definitions . . . . .	2
2. Introduction . . . . .	4
3. Problem Overview . . . . .	4
3.1. Table Ranges . . . . .	4
3.2. Table Append . . . . .	5
3.3. Error codes . . . . .	6
3.4. Bitmap Datatype . . . . .	6
4. Protocol Update Proposal . . . . .	6
4.1. Table Ranges . . . . .	6
4.2. Table Append . . . . .	7
4.3. Error Codes . . . . .	8
4.4. Bitmap Datatype . . . . .	9
5. IANA Considerations . . . . .	9
6. Security Considerations . . . . .	9
7. References . . . . .	9
7.1. Normative References . . . . .	9
7.2. Informative References . . . . .	10
Author's Address . . . . .	10

## 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 reiterates the terminology defined by the ForCES architecture in various documents for the sake of 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.

ForCES Protocol - Protocol that runs in the Fp reference points in the ForCES Framework [RFC3746].

ForCES Protocol Layer (ForCES PL) - A layer in the ForCES protocol architecture that defines the ForCES protocol messages, the protocol state transfer scheme, and the ForCES protocol architecture itself as defined in the ForCES Protocol Specification [RFC5810].

ForCES Protocol Transport Mapping Layer (ForCES TML) - A layer in ForCES protocol architecture that uses the capabilities of existing transport protocols to specifically address protocol message transportation issues, such as how the protocol messages are mapped to different transport media (like TCP, IP, ATM, Ethernet, etc.), and how to achieve and implement reliability, ordering, etc. the ForCES SCTP TML [RFC5811] describes a TML that is mandated for ForCES.

## 2. Introduction

Experience in implementing and deploying ForCES architecture has demonstrated need for a few small extensions both to ease programmability and to improve wire efficiency of some transactions. This document describes a few extensions to the ForCES Protocol Specification [RFC5810] semantics to achieve that end goal.

This document describes and justifies the need for 4 small extensions which are backward compatible.

1. A table range operation to allow a controller or control application to request or delete an arbitrary range of table rows.
2. A table append operation to allow a controller to add a new table row using the next available table index.
3. Improved Error codes returned to the controller (or control application) to improve granularity of existing defined error codes.
4. Optimization to packing and addressing commonly used bitmap structure.

## 3. Problem Overview

In this section we present sample use cases to illustrate the challenge being addressed.

### 3.1. Table Ranges

Consider, for the sake of illustration, an FE table with 1 million reasonably sized table rows which are sparsely populated.

ForCES GET requests sent from a controller (or control app) are prepended with a path to a component and sent to the FE. In the case of indexed tables, the component path can either be to a table or a table row index. A control application attempting to retrieve the

first 2000 table rows appearing between row indices 23 and 10023 can achieve its goal in one of:

- o Dump the whole table and filter for the needed 2000 table rows.
- o Send upto 10000 ForCES PL requests with monotonically incrementing indices and stop when the needed 2000 entries are retrieved.
- o Use ForCES batching to send fewer large messages (several path requests at a time with incrementing indices until you hit the require number of entries).

All of these approaches are programmatically (from an application point of view) unfriendly, tedious, and are seen as abuse of both compute and bandwidth resources.

### 3.2. Table Append

For the sake of illustration, assume that a newly spawned controller application wishes to install a table row but it has no apriori knowledge of which table index to use.

ForCES allows a controller/control app to request for the next available table index as demonstrated in (Figure 1) (refer to [RFC5810] section 4.8.2 for details of table properties).

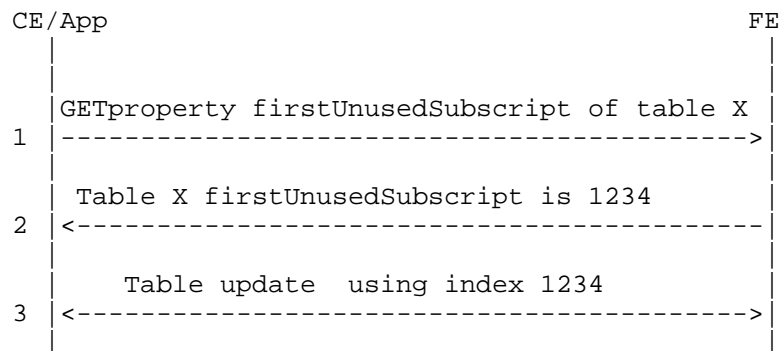


Figure 1: ForCES table property request

The problem with the above setup is the application requires one roundtrip time to figure out the index to insert into. Moreover, depending on implementation (and in presence of multiple control applications):

1. there is no guarantee that the next available subscript in the above example would stay at 1234 at the moment an application chooses to do the update; this will entirely depend on implementation at the FE and/or available holes in the table.
2. In case of multiple apps wishing to insert rows to the same table concurrently, all contending apps will be returned the same value for unused subscript; however, if all the contending apps try to insert at the same time, only the first one to reach the FE row will succeed. A solution involving a reservation mechanism to ask for an index will contribute complexity.

We conclude that even in the best case scenario, if the application wishes to insert more than one entry, it will have to incur the roundtrip time for every to-be-inserted table row. This greatly affects table add latencies and update rates.

### 3.3. Error codes

[RFC5810] has defined a generic set of error codes that are to be returned to the CE from an FE. Deployment experience has shown that it would be useful to have more fine grained error codes. As an example, the error code E\_NOT\_SUPPORTED could be mapped to many FE error source possibilities that need to be then interpreted by the caller based on some understanding of the nature of the sent request. This makes debugging more time consuming.

### 3.4. Bitmap Datatype

TBA

## 4. Protocol Update Proposal

This section describes proposals to update the protocol for issues discussed in Section 3

### 4.1. Table Ranges

We propose to add a Table-range TLV (type ID 0x117) that will be associated with the PATH-DATA TLV in the same manner the KEYINFO-TLV is.

```
OPER = GET
PATH-DATA:
  flags = F_SELTABRANGE, IDCount = 2, IDs = {1,6}
  TABLERANGE-TLV = {11,23}
```

Figure 2: ForCES table range request



Figure 2 illustrates a GET request for a table range for rows 11 to 23 of a table with component path of 1/6.

Path flag of F\_SELTABRANGE (0x2 i.e bit 1, where bit 0 is F\_SELKEY as defined in RFC 5810) is set to indicate the presence of the Table-range TLV. The pathflag bit F\_SELTABRANGE can only be used in a GET and is mutually exclusive with F\_SELKEY. The FE MUST enforce those constraints and reject a request with an error code of E\_INVALID\_FLAGS with an english description of what the problem is (refer to Section 4.3).

The Table-range TLV contents constitute:

- o A 32 bit start index. An index of 0 implies the beginning of the table row.
- o A 32 bit end index. A value of 0xFFFFFFFFFFFFFFFF implies the last entry. XXX: Do we need to define the "end wildcard"?

The response for a table range query will either be:

- o The requested table data returned (when at least one referenced row is available); in such a case, a response with a path pointing to the table and whose data content contain the row(s) will be sent to the CE. The data content MUST be encapsulated in sparsedata TLV. The sparse data TLV content will have the "I" (in ILV) for each table row indicating the table indices.
- o A result TLV when:
  - \* data is absent where the result code of E\_NOT\_SUPPORTED (typically returned in current implementations when accessing an empty table entry) with an english message describing the nature of the error (refer to Section 4.3).
  - \* When both a path key and path table range are reflected on the the pathflags, an error code of E\_INVALID\_FLAGS with an english message describing the nature of the error (refer to Section 4.3).
  - \* other standard ForCES errors (such as ACL constraints trying to retrieve contents of an unreadable table), accessing unknown components etc.

#### 4.2. Table Append

We propose using a path flag, F\_TABAPPEND(0x4, bit 2) to achieve this goal.

When a CE application wishes to append to the table, it will set the path to a desired table index and set the path flag to F\_TABAPPEND. The FE will first attempt to use the specified index and when unsuccessful will use an available table row index.

On success or failure to insert the table row, a result TLV will be returned with the appropriate code. Alternatively a the new EXTENDED-RESULT-TLV (refer to Section 4.3) maybe returned. The path of the response will contain the table row index where the table row was inserted (which the application can then learn).

When successful, an E\_SUCCESS return code is sent back to the CE.

Upon failure to append the table row, an appropriate error code is sent back to the CE.

#### 4.3. Error Codes

We propose a new TLV, EXTENDED-RESULT-TLV (0x118) that will carry both a result code as currently specified but also a string[N] cause. This is illustrated in Figure 3.

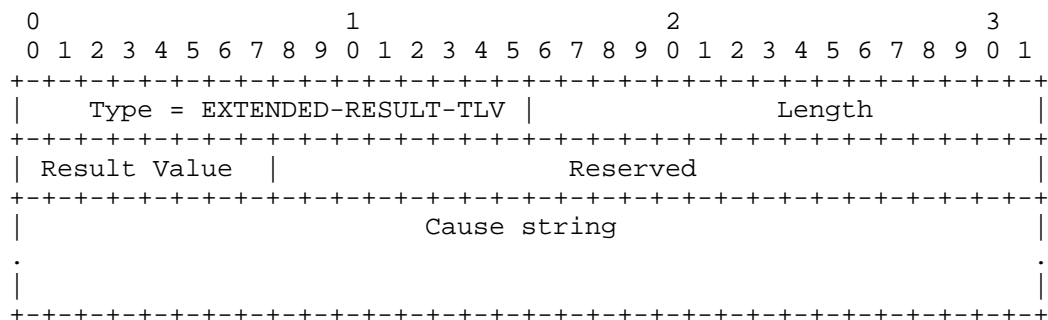


Figure 3: Extended Result TLV

- o Like all other ForCES TLVs, the Extended Result TLV is expected to be 32 bit aligned.
- o The Result Value is derived from the same current namespace as specified in RFC 5810, section 7.1.7.
- o It is recommended that the maximum size of the cause string should not exceed 32 bytes. We do not propose the cause string be standardized.

XXX: Backward compatibility may require that we add a FEPO capability to advertise ability to do extended results so that the CE is able to interpret the results.

#### 4.4. Bitmap Datatype

TBA

#### 5. IANA Considerations

This document registers two new top Level TLVs and two new path flags.

The following new TLVs are defined:

- o Table-range TLV (type ID 0x117)
- o EXTENDED-RESULT-TLV (type ID 0x118)

The following new path flags are defined:

- o F\_SELTABRANGE (value 0x2 i.e bit 1)
- o F\_TABAPPEND (value 0x4 i.e bit 2)

#### 6. Security Considerations

TBD

#### 7. References

##### 7.1. Normative References

- [RFC3746] Yang, L., Dantu, R., Anderson, T., and R. Gopal, "Forwarding and Control Element Separation (ForCES) Framework", RFC 3746, April 2004.
- [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.
- [RFC5811] Hadi Salim, J. and K. Ogawa, "SCTP-Based Transport Mapping Layer (TML) for the Forwarding and Control Element Separation (ForCES) Protocol", RFC 5811, March 2010.

- [RFC5812] Halpern, J. and J. Hadi Salim, "Forwarding and Control Element Separation (ForCES) Forwarding Element Model", RFC 5812, March 2010.

## 7.2. Informative References

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

## Author's Address

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

Email: [hadi@mojatatu.com](mailto:hadi@mojatatu.com)

Internet Engineering Task Force  
Internet-Draft  
Intended status: Informational  
Expires: January 15, 2014

D. Joachimpillai  
Verizon  
J. Hadi Salim  
Mojatatu Networks  
July 14, 2013

ForCES Inter-FE LFB  
draft-joachimpillai-forces-interfelfb-02

Abstract

Forwarding and Control Element Separation (ForCES) defines an architectural framework and associated protocols to standardize information exchange between the control plane and the forwarding plane in a ForCES Network Element (ForCES NE). RFC5812 has defined the ForCES Model which provides a formal way to represent the capabilities, state, and configuration of forwarding elements(FEs) within the context of the ForCES protocol. More specifically, the model describes the logical functions that are present in an FE, what capabilities these functions support, and how these functions are or can be interconnected. The control elements (CEs) can control the FEs using the ForCES model definition.

The ForCES WG charter has been extended to allow the LFB topology to be across FEs. This 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 January 15, 2014.

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 And Use Cases . . . . .	5
3.1. Basic Router . . . . .	5
3.1.1. Distributing The LFB Topology . . . . .	7
3.2. Arbitray Network Function . . . . .	8
3.2.1. Distributing The Arbitray Network Function . . . . .	8
4. Proposal Overview . . . . .	9
4.1. Inserting The Inter-FE LFB . . . . .	9
4.2. Inter-FE connectivity . . . . .	11
4.2.1. Inter-FE Ethernet connectivity . . . . .	13
4.2.1.1. Inter-FE Ethernet Connectivity Issues . . . . .	14
5. Detailed Description of the inter-FE LFB . . . . .	15
5.1. Data Handling . . . . .	16
5.1.1. Egress Processing . . . . .	17
5.1.2. Ingress Processing . . . . .	17
5.2. Metadata . . . . .	18
5.3. Components . . . . .	18
5.4. Capabilities . . . . .	18
5.5. Events . . . . .	18
5.6. Inter-FE LFB XML . . . . .	18
6. Acknowledgements . . . . .	18
7. IANA Considerations . . . . .	19
8. Security Considerations . . . . .	19
9. References . . . . .	19
9.1. Normative References . . . . .	19
9.2. Informative References . . . . .	19
Authors' Addresses . . . . .	19

## 1. Terminology and Conventions

### 1.1. Requirements Language

The key words "MUST", "MUST NOT", "REQUIRED", "SHALL", "SHALL NOT", "SHOULD", "SHOULD NOT", "RECOMMENDED", "MAY", and "OPTIONAL" in this document are to be interpreted as described in [RFC2119].

### 1.2. Definitions

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

**FE Model** - The FE model is designed to model the logical processing functions of an FE. The FE model proposed in this document includes three components; the LFB modeling of individual Logical Functional Block (LFB model), the logical interconnection between LFBs (LFB topology), and the FE-level attributes, including FE capabilities. The FE model provides the basis to define the information elements exchanged between the CE and the FE in the ForCES protocol [RFC5810].

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

**LFB Instance** - As a packet flows through an FE along a data path, it flows through one or multiple LFB instances, where each LFB is an instance of a specific LFB class. Multiple instances of the same LFB class can be present in an FE's data path. Note that we often refer to LFBs without distinguishing between an LFB class and LFB instance when we believe the implied reference is obvious for the given context.

**LFB Model** - The LFB model describes the content and structures in an LFB, plus the associated data definition. XML is used to provide a formal definition of the necessary structures for the modeling. Four types of information are defined in the LFB model. The core part of the LFB model is the LFB class definitions; the other three types of information define constructs associated with and used by the class definition. These are reusable data types, supported frame (packet) formats, and metadata.

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

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

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

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

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

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

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

## 2. Introduction

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

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



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

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

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

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

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

### 3. Problem Scope And Use Cases

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

#### 3.1. Basic Router

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

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

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

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

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

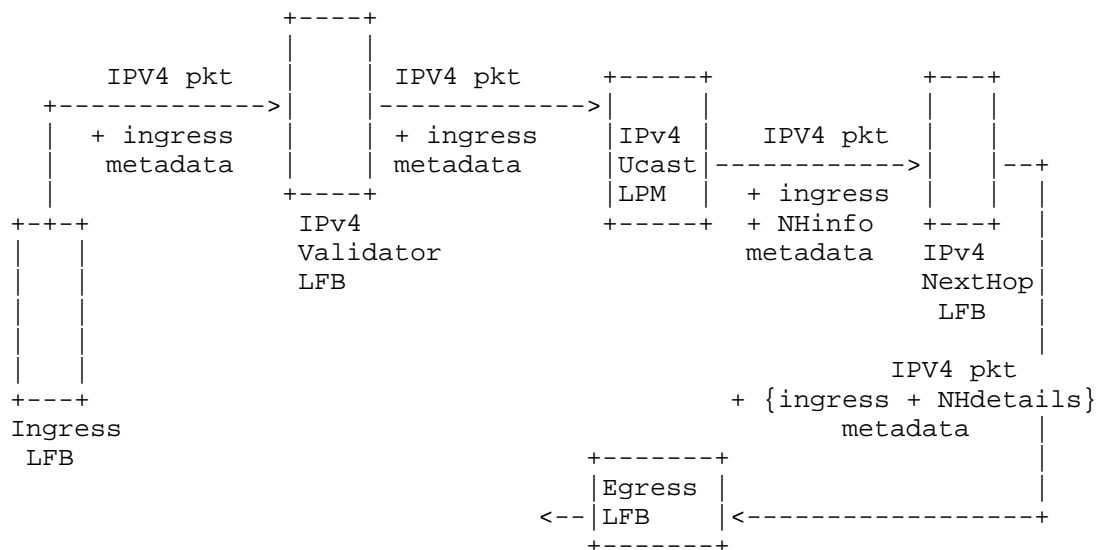


Figure 1: Basic IPV4 packet service LFB topology

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

The Nexthop LFB receives the IPV4 packet with an associated next hop

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

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

### 3.1.1. Distributing The LFB Topology

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

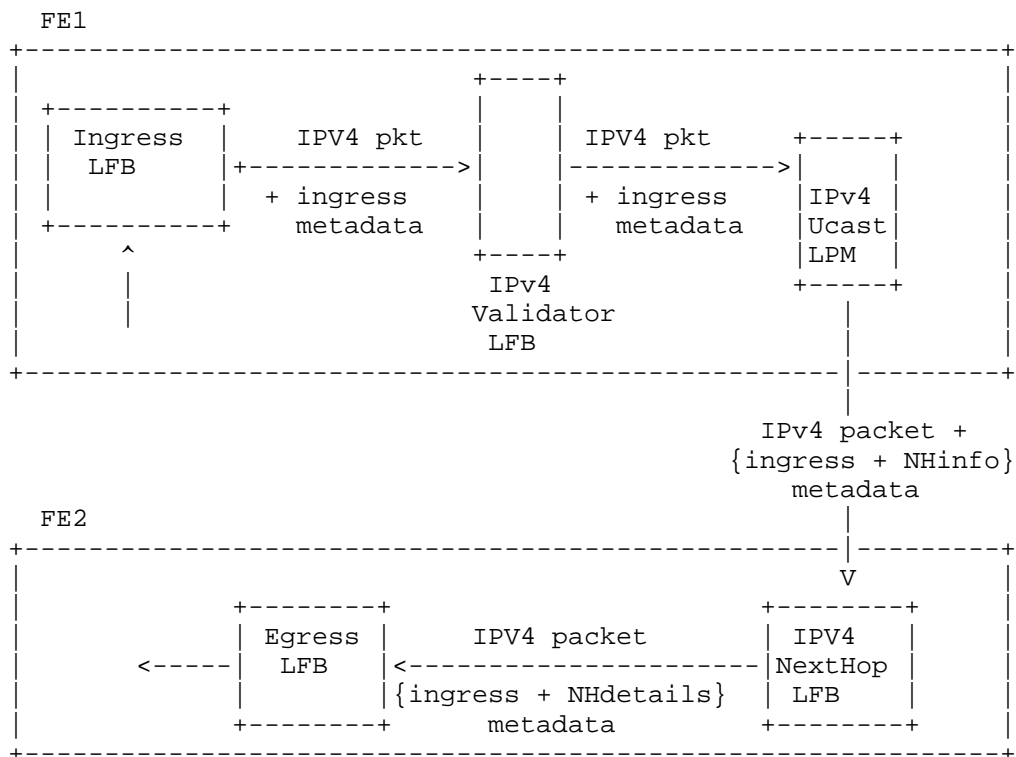


Figure 2: Split IPV4 packet service LFB topology

Some proprietary inter-connect (example Broadcom Higi over XAUI (XXX: ref needed)) maybe exist to carry both the IPV4 packet and the related metadata between the IPV4 Unicast LFB and IPV4 NextHop LFB across the two FEs.

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

### 3.2. Arbitray Network Function

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

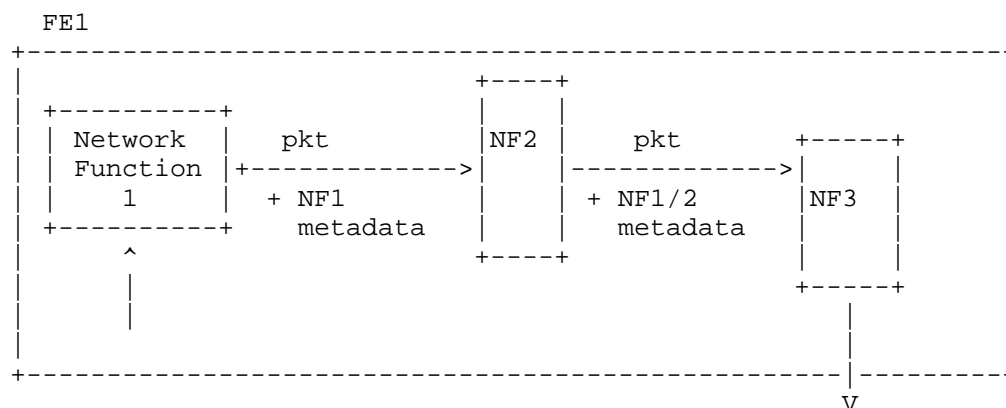


Figure 3: A Network Function Service Chain within one FE

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

#### 3.2.1. Distributing The Arbitray Network Function

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

to have the same packet service delivered to the different flows passing through.

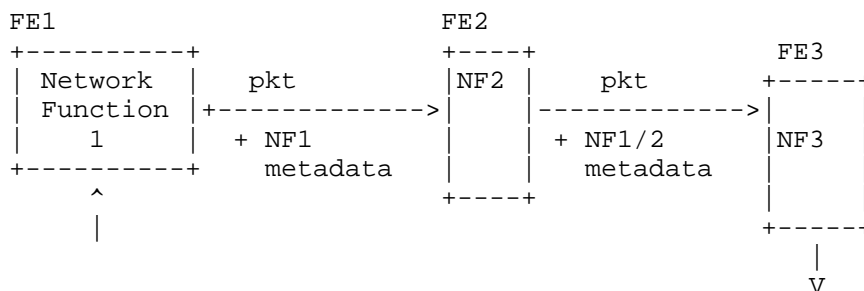


Figure 4: A Network Function Service Chain Distributed Across Multiple FEs

#### 4. Proposal Overview

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

##### 4.1. Inserting The Inter-FE LFB

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

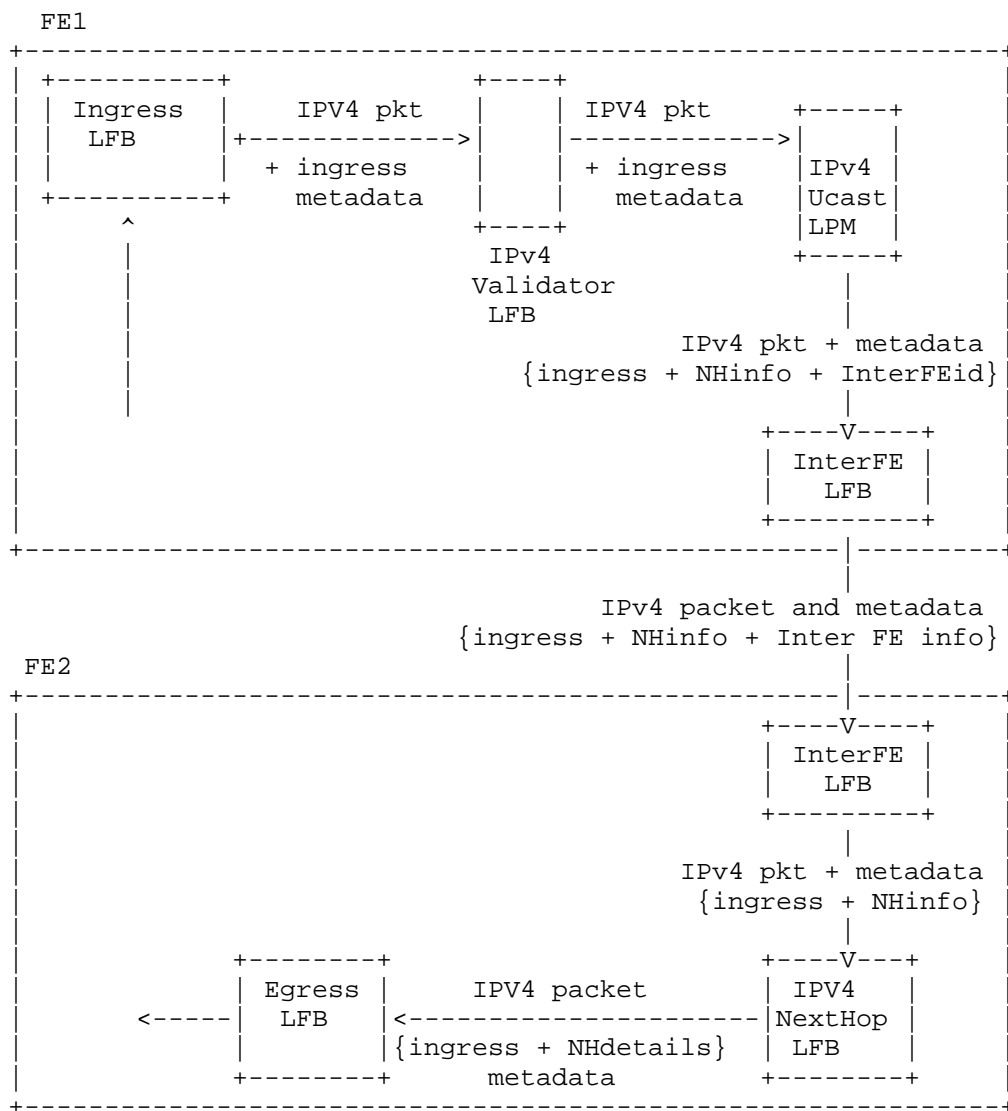


Figure 5: Split IPv4 forwarding service with Inter-FE LFB

As can be observed in Figure 5, the same details passed between IPv4 unicast LPM LFB and the IPv4 NH LFB are passed to the egress side of the Inter-FE LFB. In addition an index for the inter-FE LFB (interFEid) is passed as metadata.

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

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

On the ingress side of the inter-FE LFB the received packet and its associated details are used to decide the graph continuation i.e which FE instance is to be passed the packet 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 6) extended from the ForCES redirect packet format. We expect that for any transport mechanism used, that a description of how the different fields will be encapsulated to be explained. We provide a description of how ethernet encapsulation will be used in this case in Section 4.2.1.

```

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

```

Figure 6: Packet format suggestion

XXX: We are going to need ExceptionID-TLV to be defined. XXX: This is needed regardless of this LFB given the namespace for Exception per IANA definitions is distinct and separate from the metadata namespace.

- o The ForCES main header as described in RFC5810 is used as a fixed header to describe the Inter-FE encapsulation.
- \* The Source ID field is mapped to the originating FE and the destination ID is mapped to the destination FEID.
- \* The first 32 bits of the correlator field are used to carry the NEID. The 32-bit NEID defaults to 0.



- o The ExceptionID TLV carries one or more exception IDs within ILVs. The I in the ILV carries a globally defined exceptionID as per-ForCES specification defined by IANA. This TLV is new to ForCES and sits in the global ForCES TLV namespace.
- o The METADATA and REDIRECTDATA TLV encapsulations are taken directly from [RFC5810] section 7.9.

#### 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 6. In such a case, a description of a mapping to interpret the inter-FE details and translate into proprietary or legacy formatting would need to be defined. For any mapping towards these definitions a different document to describe the mapping, one per transport, is expected to be defined.

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.

```

*--+ Ethernet header (ethertype = XXXX)
|
+-- Main ForCES header
|
|   +---- msg type = REDIRECT
|   +---- Destination FEID
|   +---- Source FEID
|   +---- NEID -- Correlator first word
|   +---- {frag count, frag total}
|
+-- 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 7: Packet format suggestion

Notice the next 32 bits of the correlator are used for accounting of fragmentation.

#### 4.2.1.1. Inter-FE Ethernet Connectivity Issues

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

- o The frame may end up being larger than the MTU. This is to be expected in particular where one LFB instance requires assembling for example a full IPV4 message before passing it downstream to

another FE's LFB instance for further processing. There are several possible solutions:

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

We opt for the option of using the first suggestion where the sending side when fragmenting packets accounts for them as a count of total. The second 32 bit part of the ForCES correlator is split into two 16-bit fields for this activity: The first 16bit is for the fragment number and the second one is for the fragment total. As an example if there were two fragments, the first one would be: 1 of 2 and the last one 2 of 2. XXX: Outstanding question still is if we only fragment the data and not the other fields? It seems the first frame will always have all the metadata + exception TLVs and subsequent TLVs will have metadata.

## 5. Detailed Description of the inter-FE LFB

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

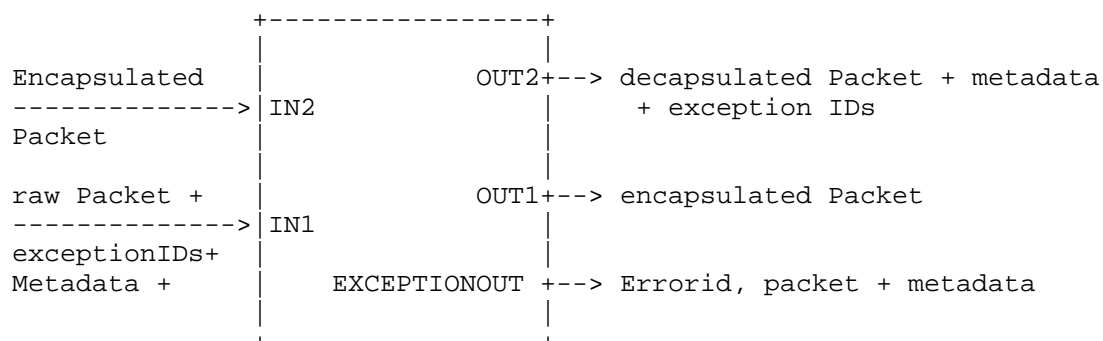


Figure 8: Inter-FE LFB

### 5.1. Data Handling

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

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

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

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

#### 5.1.1. Egress Processing

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

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

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

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

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

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

#### 5.1.2. Ingress Processing

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

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

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

XXX: We need to describe the fragmentation handling after a WG discussion.

## 5.2. Metadata

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

## 5.3. Components

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

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

## 5.4. 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

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

## 7. IANA Considerations

This memo includes two requests to IANA. One for InterFE Metaid and another for the ExceptionID-TLV. XXX: ExceptionID-TLV is needed regardless of this document, so may need to be requested separately (mayber as part of the protocol extension).

## 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

Damascane 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](mailto:hadi@mojatatu.com)



