

ForCES
Internet-Draft
Expires: September 6, 2006

J. Halpern
Self
March 5, 2006

A base Library for use with the ForCES Protocol and Model
draft-halpern-forces-lfblibrary-base-01

Status of this Memo

By submitting this Internet-Draft, each author represents that any applicable patent or other IPR claims of which he or she is aware have been or will be disclosed, and any of which he or she becomes aware will be disclosed, in accordance with [Section 6 of BCP 79](#).

Internet-Drafts are working documents of the Internet Engineering Task Force (IETF), its areas, and its working groups. Note that other groups may also distribute working documents as Internet-Drafts.

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."

The list of current Internet-Drafts can be accessed at <http://www.ietf.org/ietf/lid-abstracts.txt>.

The list of Internet-Draft Shadow Directories can be accessed at <http://www.ietf.org/shadow.html>.

This Internet-Draft will expire on September 6, 2006.

Copyright Notice

Copyright (C) The Internet Society (2006).

Abstract

The Forwarding and Control Element Separation (ForCES) working group is defining a protocol to allow a Control Element (CE) to control the behavior of a Forwarding Element (FE). The manipulations used by this protocol operate in terms of adjustments to Logical Function Blocks (LFBs) whose structure is defined by a model RFC produced by the working group. In order to build an actual solution using this protocol, there needs to be a set of Logical Function Block definitions that can be instantiated by FEs and controlled by CEs. This document provides an initial set of such definitions. It is

Internet-Draft

Forces LFB Base Library

March 2006

anticipated that additional defining documents will be produced over time.

Table of Contents

1.	Introduction	3
2.	Requirements notation	3
3.	Base Definitions	3
4.	Connectivity LFBs	4
4.1.	Generic Connectivity LFB	4
4.2.	Redirect LFB	6
4.3.	taggedInterface	8
5.	Packet Validation and Manipulation LFBs	8
5.1.	IPv4 Validator LFB	8
5.2.	IPv6 Validator LFB	9
5.3.	Meta-Data marker	11
5.4.	Packet Trimmer	12
5.5.	IPv4 outbound updater	13
5.6.	IPv6 outbound updater	13
5.7.	Duplicator	13
6.	Classifier LFBs	14
6.1.	Classifier Data Types	15
6.2.	ArbitraryClassifierLfb	21
6.3.	LPMClassifier	24
6.4.	Next Hop Applicator	24
7.	Packet Control LFBs	24
7.1.	ARPOutRequestLFB	25
7.2.	ARPinMessageLFB	25
7.3.	ICMPLFB	25
8.	Queue and Scheduler LFBs	25
8.1.	Scheduler	26
8.2.	Queue	26
9.	Acknowledgements	26
10.	Contributors	26
11.	IANA Considerations	26
12.	Security Considerations	26
13.	References	27
13.1.	Normative References	27
13.2.	Informative References	27
	Author's Address	28
	Intellectual Property and Copyright Statements	29

1. Introduction

The ForCES protocol Protocol [2] defines a protocol by which Control Elements (CEs) communicate with and control the behavior of Forwarding Elements (FEs). That control is expressed in terms of manipulations of attributes of Logical Function Blocks (LFBs). The structure and abstract semantics of LFBs is defined in Model [3]. That document also defines a single LFB Class for gaining access to FE properties including the set of LFBs and their interconnection. The Protocol [2] document defines an LFB class for manipulating the protocol properties of the FE.

In order for the protocol to be useful to control any behavior, there must be a set of LFB class definitions for the LFBs which provide that behavior. This document provides an initial set of such definitions. While this document is intended to provide an initially sufficient set of such classes, it is expected that other definitions will be developed over time, and documented in other RFCs.

[Section 3](#) provides a set of definitions, in an LFBLibrary wrapper that does not provide any classes. These are then used in each subsequent definition by the statement:

```
<load library="Base"/>
```

Following that are sections containing definitions of LFB classes. They are grouped for convenience. While there is some explanatory text in each section, the primary semantics are explained in description clauses in the LFB Class definition so as to ensure the description is available in any context that uses the definition.

[Editor's note: Most of these class definitions are completely blank. A few have been filled in to provide starting ideas to contributors.]

[2.](#) Requirements notation

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 [\[1\]](#).

[3.](#) Base Definitions

This section provides a base set of LFB frame, data type, and meta data definitions for use by all any LFB Class definitions (in this or

other documents. This section provides no actual LFB Class definitions.

```
<LFBLibrary>
  <frameDefs>
    <frameDef>
      <name>IPv4Frame</name>
      <synopsis>A frame containing an IPv4 packet.</synopsis>
    </frameDef>
    <frameDef>
      <name>IPv6Frame</name>
      <synopsis>A frame containing an IPv6 packet.</synopsis>
    </frameDef>
    <frameDef>
      <name>taggedFrame</name>
      <synopsis>A frame of any type with associated metadata.</synopsis>
    </frameDef>
    <frameDef>
      <name>metaDataFrame</name>
      <synopsis>
        A frame consisting only of meta data, with no packet.
      </synopsis>
    </frameDef>
  </frameDefs>
  <dataTypeDefs>
</dataTypeDefs>
  <metadataDefs>
</metadataDefs>
```

</LFBLibrary>

4. Connectivity LFBs

This section provides LFB class definitions for LFBs which provide connectivity between the FE and the rest of the world.

4.1. Generic Connectivity LFB

This section provides the LFB Class definition for the generic connectivity LFB. This LFB is intended to provide media and encapsulation oriented capabilities such as one might associated with an interface. It only captures those properties which relate to its function in the data flow. (So, for example, it does not provide for the IP address associated with this interface, or even an indication as to whether there is such an address.)

Halpern

Expires September 6, 2006

[Page 4]

Internet-Draft

Forces LFB Base Library

March 2006

```
<LFBLibrary provides="GenericConnectivityLFB">
  <load library="Base"/>
  <LFBClassDefs>
    <LFBClassDef LFBClassID="0x00010001">
      <name>GenericConnectivityLFB</name>
      <synopsis>
        An LFB Class for providing connectivity between an FE and
        communications media.
      </synopsis>
      <version>0.0</version>
      <description>
        This LFB Class provides a generic basis for representing
        connectivity between the FE and the outside world.

        The LFB has one or more ports for packets that the FE
        processing logic is forwrding for transmission by this
        Connectivity LFB. It has one or more ports for packets
        that the Connectivity LFB has received and is handing to
        the FE processing logic.

        Multiple ports for handline packets are supported so that
        protocol specific encapsulation and demultiplexing can be
```

provided by this LFB.

This LFB also has ports for sending packets to lower layer Connectivity LFBs and receiving packets from such lower layer Connectivity LFBs. This enables support for the processing components of interface stacks, such as PPP over Ethernet or Ethernet over MPLS.

For packets arriving from Media or lower layer connectivity, this LFB will perform appropriate media validation, then remove media specific headers, and place the relevant information in meta-data. For ethernet, the Source MAC would be in meta-data. For Frame Relay or ATM, a circuit identifier would be in meta-data. For Ethernet with VLANs, this meta-data would indicate which VLAN the packet came from.

For packets to be transmitted, meta-data indicating the destination (destination MAC or outgoing circuit, etc.) is required.

This LFB will also include statistical attributes such as the number of octets and packets sent and received, the number of various input and output errors, etc.

```
</description>
</LFBClassDef>
</LFBClassDefs>
```

```
</LFBLibrary>
```

[4.2.](#) Redirect LFB

This class definition provides for the function of sending and receiving data packets between the CE and the FE. Such data packets are accompanied by meta-data which assists the receiver processing of the packet. This LFB is implicitly tied to the protocol machinery for redirecting packets.

There may be multiple Redirect LFBs in the LFB topology. For packets from the CE to the FE, as described in Protocol [\[2\]](#) the correct LFB to handle the packet is determined by the instance ID in the redirect message. In the direction from the FE to the CE, the source instance

ID indicates which LFB is sending the packet. Redirect Source or Sink LFBs may be instantiated by simply not connecting the input or output ports of the LFB instance to any other portion of the topology.

```
<LFBLibrary provides="RedirectLFB">
  <load library="Base"/>
  <LFBClassDefs>
    <LFBClassDef LFBClassID="0x00010002">
      <name>RedirectLFB</name>
      <synopsis>
        An LFB Class definition for exchanging data packets
        between the FE and the CE.
```

```

</synopsis>
<version>0.0</version>
<inputPorts>
  <inputPort>
    <name>RedirectToCE</name>
    <synopsis>
      Port for frames to send to the CE.
    </synopsis>
    <expectation>
      <frameExpected>
        <name>taggedFrame</name>
      </frameExpected>
    </expectation>
  </inputPort>
</inputPorts>
<outputPorts>
  <outputPort>
    <name>RedirectFromCE</name>
    <synopsis>
      Port for frames to send to the CE
    </synopsis>
    <product>
      <frameProduced>taggedFrame</frameProduced>
    </product>
  </outputPort>
</outputPorts>
<description>
  This LFB represents a point of exchange of data packets
  between the CE and the FE. Packets with meta-data are
  exchanged. It is expected that the output port of a
  RedirectLFB, if it is connected at all, will be connected
  to a meta-data redirector.
</description>
</LFBClassDef>
</LFBClassDefs>
</LFBLibrary>

```


This LFB is for use instead of a GenericConnectivity LFB for use in conjunction with media interfaces which can carry meta-data. It is in some ways similar to the RedirectLFB. It is expected that it will be used with media that are used to interconnect FEs, such as modern chassis fabrics, which can carry meta-data with packets. Unlike the Redirect LFB, it is expected that for a given fabric an FE will have only one taggedInterface LFB instance.

[5.](#) Packet Validation and Manipulation LFBs

This section provides LFBs that verify or adjust contents of packets. While one could consider the classifiers a subset of this, they are sufficiently significant that they are dealt with separately.

[5.1.](#) IPv4 Validator LFB

This LFB validates the IP version and header length fields, including verifying that the packet length is at least as long as the header indicates.

This may be placed in the data path following a Connectivity LFB, or it may be placed in the data path for packets directed towards the CE, as some routers choose not to perform extensive validation on data packets to be forwarded.

```
<LFBLibrary provides="IPv4ValidatorLFB">
  <load library="Base"/>
  <LFBClassDefs>
    <LFBClassDef LFBClassID="0x00010003">
      <name>IPv4Validator</name>
      <synopsis>
        An LFB Class definition for validates the IPv4 packet.
      </synopsis>
      <version>1.0</version>
      <inputPorts>
        <inputPort>
          <name>ValidatorIn</name>
          <synopsis>
            Normal packet input.
          </synopsis>
          <expectation>
            <frameExpected>
              <ref>IPv4</ref>
            </frameExpected>
          </expectation>
        </inputPort>
      </inputPorts>
    </LFBClassDef>
  </LFBClassDefs>
</LFBLibrary>
```

```
        </expectation>
    </inputPort>
</inputPorts>
<outputPorts>
    <outputPort>
        <name>ValidatorOut</name>
        <synopsis>
            Normal packet Output.
        </synopsis>
        <product>
            <frameProduced>
                <ref>IPv4packet</ref>
            </frameProduced>
        </product>
    </outputPort>
    <outputPort>
        <name>FailOutput</name>
        <synopsis>
            The port to send packets that do not match any entries.
        </synopsis>
        <product>
            <frameProduced>
                <ref>taggedFrame</ref>
            </frameProduced>
            <metadataProduced>
                <ref>errorid</ref>
            </metadataProduced>
        </product>
    </outputPort>
</outputPorts>
<description>
    This LFB validates the IP version and header length
    fields, including verifying that the packet length
    is at least as long as the header indicates.
</description>
</LFBClassDef>
</LFBClassDefs>
</LFBLibrary>
```

[5.2.](#) IPv6 Validator LFB

This LFB validates the IP version and header length fields, including verifying that the packet length is at least as long as the header indicates.

This may be placed in the data path following a Connectivity LFB, or it may be placed in the data path for packets directed towards the

CE, as some routers choose not to perform extensive validation on data packets to be forwarded.

```
<LFBLibrary provides="IPv6ValidLFB">
  <load library="Base"/>
  <metadataDefs>
    <metadataDef>
      <name>ErrorId</name>
      <synopsis>Error Type.</synopsis>
      <metadataID>11</metadataID>
      <atomic>
        <baseType>int32</baseType>
        <specialValues>
          <specialValue value="0x00030001">
            <name>WrongIpVersion</name>
            <synopsis>the IP version wrong</synopsis>
          </specialValue>
          <specialValue value="0x00030002">
            <name>WrongLength</name>
            <synopsis>
              the packet length is not as long as
              the header indicates
            </synopsis>
          </specialValue>
          <specialValue value="0x000300FF">
            <name>otherError</name>
            <synopsis>The errors we not defined now</synopsis>
          </specialValue>
        </specialValues>
      </atomic>
    </metadataDef>
  </metadataDefs>
  <LFBClassDefs>
    <LFBClassDef LFBClassID="0x00010004">
      <name>IPv6Validator</name>
      <synopsis>
        An LFB Class definition for validates the IPv6 packet.
      </synopsis>
```

```

<version>1.0</version>
<inputPorts>
  <inputPort>
    <name>ValidatorIn</name>
    <synopsis>
      Normal packet input.
    </synopsis>
    <expectation>
      <frameExpected>

```

```

      <ref>IPv6</ref>
    </frameExpected>
  </expectation>
</inputPort>
</inputPorts>
<outputPorts>
  <outputPort>
    <name>ValidatorOut</name>
    <synopsis>
      Normal packet Output.
    </synopsis>
    <product>
      <frameProduced>
        <ref>IPv6packet</ref>
      </frameProduced>
    </product>
  </outputPort>
  <outputPort>
    <name>FailOutput</name>
    <synopsis>
      The port to send packets that do not match any entries.
    </synopsis>
    <product>
      <frameProduced>
        <ref>taggedFrame</ref>
      </frameProduced>
      <metadataProduced>
        <ref>errorid</ref>
      </metadataProduced>
    </product>
  </outputPort>
</outputPorts>

```

```

    <description>
      This LFB validates the IP version and header length
      fields, including verifying that the packet length
      is at least as long as the header indicates.
    </description>
  </LFBClassDef>
</LFBClassDefs>
</LFBLibrary>

```

[5.3.](#) Meta-Data marker

It is sometimes necessary to move information from the packet to meta-data, or from one meta-data field to another. This LFB class provides that capability. It consists of a series of processing instructions. Each instruction identifies either a meta-data element,

Halpern

Expires September 6, 2006

[Page 11]

Internet-Draft

Forces LFB Base Library

March 2006

a named packet field, or a portion of the packet identified by offset and length. The instruction also indicates what meta-data element to copy the selected data into. The target field is identified using the same data types used for the matcher target identification.

[5.4.](#) Packet Trimmer

It is sometimes necessary to remove data from the front of a packet. This LFB class provides that capability.

```

<LFBLibrary provides="PacketTrimmer">
  <load library="Base"/>
  <LFBClassDefs>
    <LFBClassDef LFBClassID="0x00010004">
      <name>PacketTrimmer</name>
      <synopsis>
        LFB removes data from the front of a packet.
      </synopsis>
      <version>1.0</version>
      <inputPorts>
        <inputPort>
          <name>PacketIn</name>
          <synopsis>
            Normal packet input.
          </synopsis>
        </inputPort>
      </inputPorts>
    </LFBClassDef>
  </LFBClassDefs>
</LFBLibrary>

```

```

    </synopsis>
    <expectation>
      <frameExpected>
        <ref>Packet</ref>
      </frameExpected>
    </expectation>
  </inputPort>
</inputPorts>
<outputPorts>
  <outputPort>
    <name>PacketOut</name>
    <synopsis>
      Normal packet Output.
    </synopsis>
    <product>
      <frameProduced>
        <ref>Packet</ref>
      </frameProduced>
    </product>
  </outputPort>
  <outputPort>
    <name>FailOut</name>
    <synopsis>

```

```

    For packets without enough bytes to remove
  </synopsis>
  <product>
    <frameProduced>
      <ref>Packet</ref>
    </frameProduced>
  </product>
</outputPort>
</outputPorts>
<attributes>
  <attribute access="read-write" elementID="1">
    <name>TrimLength</name>
    <synopsis>amount to trim from each packet</synopsis>
    <typeRef>uint32</typeRef>
  </attribute>
</attributes>
</LFBClassDef>
</LFBClassDefs>

```

</LFBLibrary>

[5.5.](#) IPv4 outbound updater

This LFB updates the TTL and header checksum in a packet to be sent by the FE. The header checksum update is performed by modification, so that erroneous checksums are still erroneous.

[5.6.](#) IPv6 outbound updater

This LFB updates the TTL and header checksum in a packet to be sent by the FE. The header checksum update is performed by modification, so that erroneous checksums are still erroneous.

[5.7.](#) Duplicator

A duplicator LFB has one input port and multiple output ports. Any packet arriving on an input port is copied so as to be sent on all output ports.

```
<LFBLibrary provides="Duplicator">
  <load library="Base"/>
  <LFBClassDefs>
    <LFBClassDef LFBClassID="0x00010007">
      <name>Duplicator</name>
      <synopsis>
        An LFB Class definition for packet duplicator LFB.
        Any packet received on an input port is
        logically copied and sent to all output ports.
      </synopsis>
      <version>1.0</version>
```

```

<inputPorts>
  <inputPort>
    <name>PacketIn</name>
    <synopsis>
      Normal packet input.
    </synopsis>
    <expectation>
      <frameExpected>
        <ref>IPv4</ref>
        <ref>IPv6</ref>
      </frameExpected>
    </expectation>
  </inputPort>
</inputPorts>
<outputPorts>
  <outputPort group="yes">
    <name>PacketOut</name>
    <synopsis>Normal packet output port group</synopsis>
    <product>
      <frameProduced>
        <ref>IPv4</ref>
        <ref>IPv6</ref>
      </frameProduced>
    </product>
  </outputPort>
</outputPorts>
</LFBClassDef>
</LFBClassDefs>
</LFBLibrary>

```

6. Classifier LFBs

This section provides the classifier LFBs. It also includes a set of data type definitions for use by classifiers.

Currently, two classifiers are defined here. One has the ability to classify a packet based on combinations of meta data and packet contents. It has the ability to add meta-data, and to select an egress port. It may be useful to define classes with only a subset

of these capabilities. The other does an longest prefix match (LPM) lookup of the value provided in the "target" meta-data item.

[6.1.](#) Classifier Data Types

These data definitions belong in a `dataTypeDefs` element in some `LFBLibrary`.

These data definitions are built around a simplistic classifier model. The classifier consists of a sequence of test-action pairs. The each test consists of an optional input port number condition followed by a sequence of match conditions. A test is considered passed if all of the match conditions succeed. The classifier conceptually functions by applying success tests until one succeeds.

First, there is the definition of the scalar for the target of a match.

```
<dataTypeDef>
  <name>MatchTargetType</name>
  <synopsis>
    Indicator for the kind of field to be matched by this
    entry in a classifier.
  </synopsis>
  <atomic>
    <baseType>uint8</baseType>
    <specialValues>
      <specialValue value="0">
        <name>MatchNone</name>
        <synopsis>A matcher against no field</synopsis>
      </specialValue>
      <specialValue value="1">
        <name>MatchMetaData</name>
        <synopsis>A matcher against a metadata item</synopsis>
      </specialValue>
      <specialValue value="2">
        <name>MatchPacketField</name>
        <synopsis>
          A matcher that works against an identified packet field.
        </synopsis>
      </specialValue value="3">
        <name>MatchOffsetLength</name>
        <synopsis>
          The match target is a specified portion of the packet.
        </synopsis>
      </specialValue>
    </specialValues>
  </atomic>
</dataTypeDef>
```

Then there is the data type definition for the identifier of the target of the match.

```
<dataTypeDef>
  <name>MatchTargetIdentifier</name>
  <synopsis>
    Identify the specific target of a match condition.
  </synopsis>
  <union>
    <element elementID="1">
      <name>MetaDataID</name>
      <synopsis>The ID of a metadata item</synopsis>
      <typeRef>uint32</typeRef>
    </element>
    <element elementID="2">
      <name>packetFieldID</name>
      <synopsis>
        The identifier for a packet Field, such as SA, DA,
        Protocol, SPort, DPort, etc. These identifiers allow
        references to fields with variable amounts before them.
      </synopsis>
      <typeRef>uint32</typeRef>
    </element>
    <element elementID="3">
      <name>OffsetLengthPacketField</name>
      <synopsis>
        A field in the packet identified by its offset and
        length in bits. This does not allow for matching fields
        whose position depends upon earlier field sizes.
      </synopsis>
      <struct>
        <element elementID="1">
          <name>fieldOffset</name>
          <synopsis>
            The offset in bits from the start of the packet to the
            start of the field.
          </synopsis>
          <typeRef>uint32</typeRef>
        </element>
        <element elementID="2">
          <name>fieldLength</name>
          <synopsis>The length of the field, in bits</synopsis>
          <typeRef>uint32</typeRef>
        </element>
      </struct>
    </element>
  </union>
</dataTypeDef>
```

```

        </element>
    </struct>
</element>
</union>
</dataTypeDefs>

```

Then there is the representation of the match condition. First we

provide the structure definition for a match condition, and then the enumeration that defines the various conditions. This ordering is for readability.

The conditions use bitfields, which are represented as octet strings of length up to 16 bytes, along with a length providing the actual meaningful length in bits. The model could be enhanced to provide a base type for variable length bit strings.

```

<dataTypeDef>
  <name>MatchBitString</name>
  <synopsis>A bit string for use in a match condition.</synopsis>
  <struct>
    <element elementID="1">
      <name>MatchBits</name>
      <synopsis>The bits to match</synopsis>
      <typeRef>OctetString[16]</typeRef>
    </element>
    <element elementID="2">
      <name>MatchLength</name>
      <synopsis>The number of bits to match</synopsis>
      <typeRef>uint8</typeRef>
    </element>
  </struct>
</dataTypeDef>

<dataTypeDef>
  <name>MatchCondition</name>
  <synopsis>
    structure for a single condition to be applied.
  </synopsis>
  <struct>

```

```

<element elementID="1">
  <name>TargetType</name>
  <synopsis>The category of target to match</synopsis>
  <typeRef>MatchTargetType</typeRef>
</element>
<element elementID="2">
  <name>TargetID</name>
  <synopsis>The specific target to compare</synopsis>
  <typeRef>MatchTargetIdentifier</typeRef>
</element>
<element elementID="3">
  <name>MatchType</name>
  <synopsis>The kind of match to apply.</synopsis>
  <typeRef>MatchConditionType</typeRef>
</element>

```

Halpern

Expires September 6, 2006

[Page 18]

Internet-Draft

Forces LFB Base Library

March 2006

```

<element elementID="4">
  <name>MatchParamOne</name>
  <synopsis>The first parameter for the match</synopsis>
  <optional/>
  <typeRef>MatchBitString</typeRef>
<element elementID="5">
  <name>MatchParamTwo</name>
  <synopsis>The second parameter for the match</synopsis>
  <optional/>
  <typeRef>MatchBitString</typeRef>
</element>
</struct>
</dataTypeDef>

```

The enumeration describes the match types, and how it interacts with the structure of a match condition. There may be more conditions here than we need.

```

<dataTypeDef>
  <name>MatchConditionType</name>
  <synopsis>
    Indicator for the kind of match condition to be applied.
  </synopsis>
  <atomic>

```

```

<baseType>uint8</baseType>
<specialValues>
  <specialValue value="0">
    <name>MatchNone</name>
    <synopsis>A matcher which always fails</synopsis>
  </specialValue>
  <specialValue value="1">
    <name>MatchExact</name>
    <synopsis>
      The target and the match value must be the same, with no
      padding. Only the first value of the match condition is
      used. The first match value must be occur.
    </synopsis>
  </specialValue>
  <specialValue value="2">
    <name>MatchLeft</name>
    <synopsis>
      The target must begin with the first match value.
      If there is a second match value, the remainder of the
      target must match repeated occurrences of the second
      value. Thus, this can be used to allow any terminal
      content, or specific ending pad. The first match value

```

```

      must occur.
    </synopsis>
  </specialValue>
  <specialValue value="3">
    <name>MatchRight</name>
    <synopsis>
      The target must end with the first match value.
      If there is a second match value, the preceding part
      of the target must match repeated occurrences of the
      second value. Thus, this can be used to allow any
      leading content, or specific leading fill. The first
      match value must occur.
    </synopsis>
  </specialValue>
  <specialValue value="4">
    <name>MatchRange</name>
    <synopsis>
      The match values will be considered as numbers, and
      the target must be greater than or equal to the

```

```

        first match value, and less than or equal to the
        second match value. An omitted match value means
        that end of the range is unlimited.
    </synopsis>
</specialValue>
<specialValue value="5">
    <name>MatchMaskedValue</name>
    <synopsis>
        The target the the first value are each anded with the
        second value. The match succeeds if the results of these
        and operations are identical. Both values are required.
    </synopsis>
</specialValue>
<specialValue value="6">
    <name>MatchSucceed</name>
    <synopsis>A Match which always succeeds</synopsis>
</specialValue>
</specialValues>
</atomic>
</dataTypeDef>

```

The MatchMetaData Action represents setting a piece of metadata when all of the match conditions are met. The action can set the meta-data to a specific value, or can set it to a value used by a match condition.

The two kinds of values are used, without a union, for simplicity.

The match condition value is used as that avoids the question of whether a specific field exists in the packet. It must exist for it to have matched.

```

<dataTypeDef>
    <name>MatchMetaDataAction</name>
    <synopsis>
        An action to set a metadata item to either a specific value
        or a field from the incoming meta data or packet.
    </synopsis>
    <struct>

```

```

<element elementID="1">
  <name>MetaDataToSet</name>
  <synopsis>The Meta Data Item to set</synopsis>
  <typeRef>uint32</typeRef>
</element>
<element elementID="2">
  <name>ExplicitValueToSet</name>
  <synopsis>A value to set the metadata to</synopsis>
  <optional/>
  <typeRef>OctetString[16]</typeRef>
</element>
<element elementID="3">
  <name>ValueFromCondition</name>
  <synopsis>
    This is an index into the corresponding match conditions,
    and the meta data will be set to the value that was tested
    by that condition.
  </synopsis>
  <optional/>
  <typeRef>uint32</typeRef>
</element>
</struct>
</dataTypeDef>

```

6.2. ArbitraryClassifierLfb

This is a class definition that makes use of the above types. The input is a port group, and the match conditions can include the port in their test. This allows the topology to carry some information if desired. The match conditions can select an output from the SuccessOutput output port group. If no condition matches, the packet will be sent to the FailOutput port.

```
<LFBLibrary provides="RedirectLFB">
```

```

<load library="Base"/>
<LFBClassDefs>
  <LFBClassDef LFBClassId="0x00010004">
    <name>ArbitraryClassifierLfb</name>
    <synopsis>

```



```

    A classifier which can test packet or metadata, and on that
    basis set meta-data a pick an output port.
</synopsis>
<version>0.0</version>
<inputPorts>
  <inputPort group="yes">
    <name>PacketsToClassify</name>
    <synopsis>
      The group of ports to received packets over
    </synopsis>
    <expectation>
      <frameExpected>
        <ref>taggedFrame</ref>
      </frameExpected>
<!-- no metaDataExpected item as any and all meta data is allowed -->
    </expectation>
  </inputPort>
</inputPorts>
<outputPorts>
  <outputPort group="yes">
    <name>SuccessOutput</name>
    <synopsis>
      The group of ports used by the classifier for output
      when a successful match is found.
    </synopsis>
    <product>
      <frameProduced>
        <ref>taggedFrame</ref>
      </frameProduced>
<!-- no metaDataProduced as anything can be produced -->
    </product>
  </outputPort>
  <outputPort group="no">
    <name>FailOutput</name>
    <synopsis>
      The port to send packets that do not match any entries.
    </synopsis>
    <product>
      <frameProduced>
        <ref>taggedFrame</ref>
      </frameProduced>
<!-- no metaDataProduced as anything can be produced -->
    </product>

```

```

    </outputPort>
</outputPorts>
<attributes>
  <attribute access="read-write" elementID="1">
    <name>ClassifierTable</name>
    <synopsis>
      The table of classifier entries
      Each entry is tested until one succeeds.
      Each entry contains an optional port test, an array of
      packet and meta data tests, an array of metadata actions,
      and an exit selection.
    </synopsis>
    <array type="variable-size">
      <struct>
        <element elementID="1">
          <name>InputPortTest</name>
          <synopsis>
            If present, this match will only match packets
            arriving over the specified port.
          </synopsis>
          <optional/>
          <typeRef>uint32</typeRef>
        </element>
        <element elementID="2">
          <name>TestConditions</name>
          <synopsis>The array of conditions to test</synopsis>
          <array type="variable-size">
            <typeRef>MatchCondition</typeRef>
          </array>
        </element>
        <element elementID="3">
          <name>MetaDataActions</name>
          <synopsis>
            The array of meta data modifications to make when the
            match succeeds.
          </synopsis>
          <array type="variable-size">
            <typeRef>MatchMetaDataAction</typeRef>
          </array>
        </element>
        <element elementID="4">
          <name>MatchOutputPort</name>
          <synopsis>
            The port within the success group to send packets
            which match these tests.
          </synopsis>
          <typeRef>uint32</typeRef>
        </element>
      </struct>
    </array>
  </attribute>
</attributes>

```

Internet-Draft

Forces LFB Base Library

March 2006

```
        </struct>
      </array>
    </attribute>
  </attributes>
  <capabilities>
  </capabilities>
</LFBClassDef>
</LFBClassDefs>
</LFBLibrary>
```

[6.3.](#) LPMClassifier

This takes the information in the "target" metadata item, and looks it up in its longest prefix match table. It sets the "LPMresult" meta-data item to the value associated with the best match entry. For example, the result might be a next-hop identifier.

[6.4.](#) Next Hop Applicator

This LFB class is used to apply a next hop to a packet. The next hop is identified by the NextHop meta-data. The value of that meta-data is used as an index into the next-hop table owned by this instance of this class. The table indicates a series of new meta-data items to add to the packet, and an exit port from the Success port group. If no valid next hop is found, the packet is sent to the MissingEntry port. If a valid next hop is found, but it needs resolution, the packet is sent to the NeedsResolution port, which will typically lead to a suitable ARPOutRequest LFB instance.

As part of the functioning of this LFB, one of the next hop identifiers would indicate packets to be sent to the CE. One of the outputs of this Next Hop applicator would be connected to a path leading to a Redirect LFB to handle those packets.

Note that some FEs will have restrictions in their actual implementation such that the LPM always goes against certain packet fields, and always produces a block of information rather than an identifier. Some of those restrictions can be represented by the FE Object LFB Support LFB Attribute CanOccurBefore and CanOccurAfter information.

[7.](#) Packet Control LFBs

These LFBs are related to control functions for data packets.

Halpern

Expires September 6, 2006

[Page 24]

Internet-Draft

Forces LFB Base Library

March 2006

[7.1.](#) ARPOutRequestLFB

Given a data packet and a next hop identifier, this LFB builds an ARP request and hands it off. It has an alias to point to the next hop table that is shared with the output encapsulator in the connectivity LFB and with other packet processors.

[7.2.](#) ARPInMessageLFB

This LFB is handed received ARP messages, both requests and responses. It performs table updating (using alias entries) and when necessary generates ARP responses.

[7.3.](#) ICMPLFB

This is handed a packet with meta-data indicating a problem. It determines if an ICMP message should be generated, and if so to whom it should be sent.

[8.](#) Queue and Scheduler LFBs

To build an actual forwarder, one must include some limited form of queueing and scheduling. Queues are entities which store packets. Schedulers are entities which react to the state of queues and cause packets to be emitted from queues.

The actual interaction between queues and schedulers (and their real world degree of separation) is quite complex. A very complex LFB model would be required to represent all the complexity. Additionally, there is the issue of representing the relationship between the queue and the scheduler. A simple approach has been taken in these class definitions.

A queue element consists of an input port (called InData) on which it

receives data packets, and output port (called OutData) on which it will send packets when permitted by its definition or the scheduler. Its relationship to schedulers is represented by a set of output ports (the group OutControl) and an input port (called InControl). These ports are defined to carry packets consisting only of meta-data. In fact, these ports are an abstraction, and what one might call a legal fiction. An element of the OutControl group represents the fact that a scheduler is aware of the state of that queue element. The InControl port represents the fact that one or more schedulers connected to that port are controlling that queue. There is no meta-data defined for actual exchange on these ports, as their real world realization is highly implementation dependent. To complete this picture, a schedule has a group of input ports

(Watchers) representing the connectivity to queues it is aware of, and a group of output ports (Controllers) representing control over queues. This allows for the simple case of a controller who monitors and controls a single set of queues, and more interesting cases where the control of certain queues may depend upon the state of queues which are not under the control of the scheduler.

[8.1.](#) Scheduler

This defines a base LFB class for schedulers. Schedulers have an Input Port group called Watchers for representing the queues they watch, and an Output Port group called Controllers for representing the queues they control.

[8.2.](#) Queue

Queues have a packet input, a packet output, a control input, and a group of control outputs. The control ports represent the control relationships with schedulers.

[9.](#) Acknowledgements

The ideas here are based on proposals from many people. In particular, Xiaoyi Guo has provided some of the LFB definitions included herein.

[10.](#) Contributors

The following people contributed significant portions of text to this document.

[11.](#) IANA Considerations

The ForCES working group needs to determine how LFB Class IDs will be registered. It seems likely that an IANA registry will be needed. Once that registry is established by the Model draft, this document will need to register values for the LFB classes it defines.

[12.](#) Security Considerations

These definitions if used by an FE to support ForCES create manipulable entities on the FE. Manipulation of such objects can produce almost unlimited effects on the FE. FEs should ensure that only properly authenticated ForCES protocol participants are

Halpern

Expires September 6, 2006

[Page 26]

Internet-Draft

Forces LFB Base Library

March 2006

performing such manipulations. Thus, largely, the security issues with this protocol are defined in Protocol [\[2\]](#).

[13.](#) References

[13.1.](#) Normative References

- [1] Bradner, S., "Key words for use in RFCs to Indicate Requirement Levels", [BCP 14](#), [RFC 2119](#), March 1997.
- [2] Doria, A., "I-D.ietf-forces-protocol-04.txt", 2005.
- [3] Deleganes, E., "I-D.ietf-forces-model-05.txt", 2005.

[13.2.](#) Informative References

Halpern

Expires September 6, 2006

[Page 27]

Internet-Draft

Forces LFB Base Library

March 2006

Author's Address

Joel M. Halpern
Self
P. O. Box 6049
Leesburg, VA 20178
US

Email: jmh@joelhalpern.com

Intellectual Property Statement

The IETF takes no position regarding the validity or scope of any Intellectual Property Rights or other rights that might be claimed to pertain to the implementation or use of the technology described in this document or the extent to which any license under such rights might or might not be available; nor does it represent that it has

made any independent effort to identify any such rights. Information on the procedures with respect to rights in RFC documents can be found in [BCP 78](#) and [BCP 79](#).

Copies of IPR disclosures made to the IETF Secretariat and any assurances of licenses to be made available, or the result of an attempt made to obtain a general license or permission for the use of such proprietary rights by implementers or users of this specification can be obtained from the IETF on-line IPR repository at <http://www.ietf.org/ipr>.

The IETF invites any interested party to bring to its attention any copyrights, patents or patent applications, or other proprietary rights that may cover technology that may be required to implement this standard. Please address the information to the IETF at ietf-ipr@ietf.org.

Disclaimer of Validity

This document and the information contained herein are provided on an "AS IS" basis and THE CONTRIBUTOR, THE ORGANIZATION HE/SHE REPRESENTS OR IS SPONSORED BY (IF ANY), THE INTERNET SOCIETY AND THE INTERNET ENGINEERING TASK FORCE DISCLAIM ALL WARRANTIES, EXPRESS OR IMPLIED, INCLUDING BUT NOT LIMITED TO ANY WARRANTY THAT THE USE OF THE INFORMATION HEREIN WILL NOT INFRINGE ANY RIGHTS OR ANY IMPLIED WARRANTIES OF MERCHANTABILITY OR FITNESS FOR A PARTICULAR PURPOSE.

Copyright Statement

Copyright (C) The Internet Society (2006). This document is subject to the rights, licenses and restrictions contained in [BCP 78](#), and except as set forth therein, the authors retain all their rights.

Acknowledgment

Funding for the RFC Editor function is currently provided by the Internet Society.