

Internet Engineering Task Force  
Internet-Draft  
Intended status: Informational  
Expires: January 10, 2013

E. Haleplidis  
University of Patras  
O. Cherkaoui  
University of Quebec in Montreal  
S. Hares  
Huawei  
W. Wang  
Zhejiang Gongshang University  
July 9, 2012

Forwarding and Control Element Separation (ForCES) OpenFlow Model  
Library  
draft-haleplidis-forces-openflow-lib-01

## Abstract

This document describes the OpenFlow switch in Logical Function Blocks (LFBs) used in the Forwarding and Control Element Separation (ForCES). The LFB classes are defined according to the ForCES Forwarding Element (FE) model and ForCES protocol specifications. The library includes the descriptions of the OpenFlow LFBs and the XML definitions.

## 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 10, 2013.

## Copyright Notice

Copyright (c) 2012 IETF Trust and the persons identified as the document authors. All rights reserved.

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

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

## Table of Contents

1. Introduction . . . . .	5
1.1. ForCES . . . . .	5
1.2. OpenFlow . . . . .	5
2. Terminology and Conventions . . . . .	7
2.1. Requirements Language . . . . .	7
2.2. Definitions . . . . .	7
3. OpenFlow ForCES library . . . . .	10
3.1. OpenFlow Specification . . . . .	10
3.2. ForCES-based OpenFlow Specification . . . . .	12
4. OpenFlow Base Types . . . . .	21
4.1. Data Types . . . . .	21
4.1.1. Atomic . . . . .	21
4.1.2. Compound Struct . . . . .	22
4.1.3. Compound Array . . . . .	23
4.2. Frame Types . . . . .	23
4.3. MetaData Types . . . . .	23
5. OpenFlow LFBs . . . . .	25
5.1. OpenFlowSwitch . . . . .	25
5.1.1. Data Handling . . . . .	25
5.1.2. Components . . . . .	25
5.1.3. Capabilities . . . . .	26
5.1.4. Events . . . . .	26
5.2. OFFlowTables . . . . .	26
5.2.1. Data Handling . . . . .	27
5.2.2. Components . . . . .	28
5.2.3. Capabilities . . . . .	29
5.2.4. Events . . . . .	29
5.3. OFGroupTable . . . . .	29
5.3.1. Data Handling . . . . .	29
5.3.2. Components . . . . .	29
5.3.3. Capabilities . . . . .	30
5.3.4. Events . . . . .	30
5.4. OFPort . . . . .	30
5.4.1. Data Handling . . . . .	30
5.4.2. Components . . . . .	30
5.4.3. Capabilities . . . . .	31
5.4.4. Events . . . . .	31

5.5.	OFQueue	31
5.5.1.	Data Handling	31
5.5.2.	Components	31
5.5.3.	Capabilities	32
5.5.4.	Events	32
5.6.	OFRedirectIn	32
5.6.1.	Data Handling	32
5.6.2.	Components	32
5.6.3.	Capabilities	32
5.6.4.	Events	32
5.7.	OFRedirectOut	33
5.7.1.	Data Handling	33
5.7.2.	Components	33
5.7.3.	Capabilities	33
5.7.4.	Events	33
5.8.	OFAction	33
5.8.1.	Data Handling	33
5.8.2.	Components	34
5.8.3.	Capabilities	34
5.8.4.	Events	34
5.9.	OFActionLFBs	34
5.9.1.	OFActionOutput	34
5.9.2.	OFActionSetVLANVID	35
5.9.3.	OFActionSetVLANPriority	35
5.9.4.	OFActionSetMACSource	35
5.9.5.	OFActionSetMACDestination	35
5.9.6.	OFActionSetIPSource	36
5.9.7.	OFActionSetIPDestination	36
5.9.8.	OFActionSetIPTOS	36
5.9.9.	OFActionSetIPECN	36
5.9.10.	OFActionSetTCPSource	37
5.9.11.	OFActionSetTCPDestination	37
5.9.12.	OFActionCopyTTLOut	37
5.9.13.	OFActionCopyTTLIn	38
5.9.14.	OFActionSetMPLSLabel	38
5.9.15.	OFActionSetMPLSTC	38
5.9.16.	OFActionSetMPLSTTL	38
5.9.17.	OFActionDecrementMPLSTTL	39
5.9.18.	OFActionPushVlan	39
5.9.19.	OFActionPopVLAN	39
5.9.20.	OFPushMPLSOFAction	39
5.9.21.	OPopMPLSOFAction	40
5.9.22.	OFSetQueueOFAction	40
5.9.23.	OFSetIPTTLOFAction	40
5.9.24.	OFDecrementIPTTLOFAction	40
5.9.25.	OFExperimenterOFAction	41
6.	XML for OpenFlow library	42
7.	Acknowledgements	92

8. IANA Considerations . . . . .	93
9. Security Considerations . . . . .	94
10. References . . . . .	95
10.1. Normative References . . . . .	95
10.2. Informative References . . . . .	96
Authors' Addresses . . . . .	97

## 1. Introduction

The purpose of this document is to create a library of Logical Functional Blocks that are necessary to describe an OpenFlow switch using the ForCES model. This includes DataTypes, MetaData and of course the LFBs.

Readers of this document can get a better understanding of what are the internal parts of an OpenFlow switch in a more formal approach. Additionally having a ForCES-defined OpenFlow switch allows developers to build a purely ForCES based solution that understands the OF model or even a middleware so that ForCES-implemented OpenFlow switches may be controlled by an OpenFlow controller, or a ForCES Control Element (CE) may control OpenFlow switches

### 1.1. ForCES

ForCES [RFC3746], focuses on the communication and model necessary to separate control-plane functionality such as routing protocols, signaling protocols, and admission control, from data-forwarding-plane per-packet activities, such as packet forwarding, queuing, and header editing.

The modeling of FEs is based on an abstraction using distinct Logical Functional Blocks (LFBs), which are interconnected in a directed graph, and receive, process, modify, and transmit packets along with metadata. An LFB is a block of encapsulated fine-grained operation of the forwarding plane. The ForCES model [RFC5812] additionally includes both a capability and a state model. One of the advantages of the ForCES Model is that it is independent of the actual implementation of the FE; it only provides a view of its capabilities and state that can be acted upon using the ForCES protocol. It is left to the forwarding plane developers to define how the FE functionality is represented using the model.

The ForCES protocol [RFC5810] was developed to allow the CEs to determine the capabilities of each FE expressed by the FE model, to add and remove entries, parameters, query for statistics, and register for and receive events in a scalable fashion over secure and reliable means. The strength of the ForCES protocol stems from the fact that it is agnostic of the model, as a CE can control any Forwarding Element described with the ForCES model.

### 1.2. OpenFlow

OpenFlow [OpenFlowSpec1.1] is conceptually similar to ForCES on separating the control and forwarding plane. It provides a protocol that mediates between the controller and the switch. Unlike ForCES,

the OpenFlow switch is statically defined to deal with flows and the protocol is aware of the flow components. An OpenFlow Switch consists of one or more flow tables, a group table that performs packet lookups and forwarding, and an OpenFlow channel to an external controller. A flow table is consisted of flow entries, each containing a set of match fields to match against packets, counters and instructions. The controller manages the switch via the OpenFlow protocol. Using this protocol, the controller can add, update, and delete flow and group entries.

## 2. Terminology and Conventions

### 2.1. 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].

### 2.2. Definitions

This document follows the terminology defined by ForCES related documents of RFC3654, RFC3746, RFC5810, RFC5811, RFC5812, RFC5812. The definitions are repeated below for clarity. Also additional definitions from the OpenFlow specification 1.1 [OpenFlowSpec1.1] are also included.

Control Element (CE) - A logical entity that implements the ForCES protocol and uses it to instruct one or more FEs on how to process packets. CEs handle functionality such as the execution of control and signaling protocols.

Forwarding Element (FE) - A logical entity that implements the ForCES protocol. FEs use the underlying hardware to provide per-packet processing and handling as directed/controlled by one or more CEs via the ForCES protocol.

LFB (Logical Functional Block) - The basic building block that is operated on by the ForCES protocol. The LFB is a well defined, logically separable functional block that resides in an FE and is controlled by the CE via the ForCES protocol. The LFB may reside at the FE's datapath and process packets or may be purely an FE control or configuration entity that is operated on by the CE. Note that the LFB is a functionally accurate abstraction of the FE's processing capabilities, but not a hardware-accurate representation of the FE implementation.

LFB Class and LFB Instance - LFBs are categorized by LFB Classes. An LFB Instance represents an LFB Class (or Type) existence. There may be multiple instances of the same LFB Class (or Type) in an FE. An LFB Class is represented by an LFB Class ID, and an LFB Instance is represented by an LFB Instance ID. As a result, an LFB Class ID associated with an LFB Instance ID uniquely specifies an LFB existence.

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. It defines the functionality but not how

metadata is encoded within an implementation.

LFB Components - Operational parameters of the LFBs that must be visible to the CEs are conceptualized in the FE model as the LFB components. The LFB components include, for example, flags, single-parameter arguments, complex arguments, and tables that the CE can read and/or write via the ForCES protocol (see below).

ForCES Protocol - While there may be multiple protocols used within the overall ForCES architecture, the term "ForCES protocol" and "protocol" refer to the "Fp" reference points in the ForCES framework in [RFC3746]. This protocol does not apply to CE-to-CE communication, FE-to-FE communication, or to communication between FE and CE managers. Basically, the ForCES protocol works in a master-slave mode in which FEs are slaves and CEs are masters.

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, multicast, ordering, etc. The ForCES TML specifications are detailed in separate ForCES documents, one for each TML.

Match Field - a field against which a packet is matched, including packet headers, the ingress port, and the metadata value.

Action - an operation that forwards the packet to a port or modifies the packet, such as decrementing the TTL field. Actions may be specified as part of the instruction set associated with a flow entry or in an action bucket associated with a group entry.

Flow entry - an element in a flow table used to match and process packets. It contains a set of match fields for matching packets, a set of counters to track packets, and a set of instructions to apply.

Instruction - an operation that either contains a set of actions to add to the action set, contains a list of actions to apply immediately to the packet, or modifies pipeline processing.

Flow Table - A stage of the pipeline, contains flow entries.

OpenFlow pipeline - the set of linked flow tables that provide matching, forwarding, and packet modifications in an OpenFlow switch.



Groups - a list of action buckets and some means of choosing one or more of those buckets to apply on a per-packet basis.

Group Entry - an element in a group table. It contains a group identifier to distinguish groups, a group type to define the type of the group, a set of counters to track packets, and a set of action buckets.

Action Bucket - a set of actions and associated parameters, defined for groups.

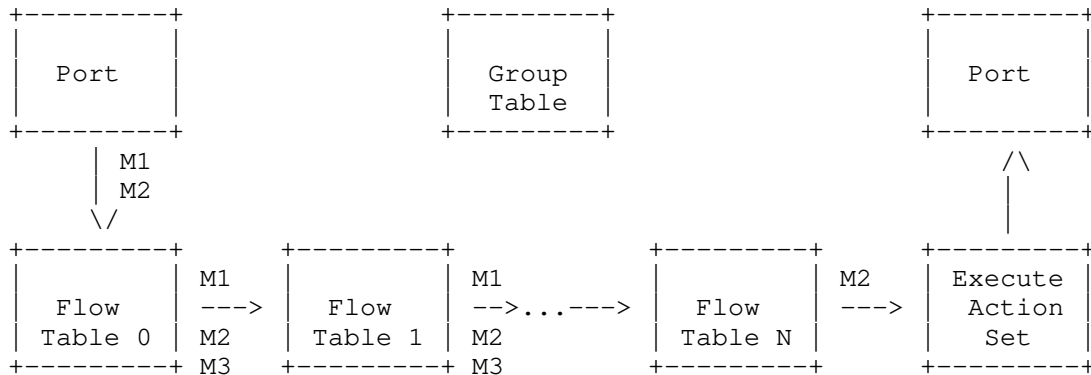
Action Set - a set of actions associated with the packet that are accumulated while the packet is processed by each table and that are executed when the instruction set instructs the packet to exit the processing pipeline.

Ports - where packets enter and exit the OpenFlow pipeline. May be a physical port, a logical port defined by the switch, or a reserved port defined by the specification.

### 3. OpenFlow ForCES library

#### 3.1. OpenFlow Specification

An OpenFlow switch as described in the OpenFlow Specification document [OpenFlowSpec1.1] appears in Figure 1



#### Legend

M1: Ingress Port

M2: Action Set{}

M3: Metadata

Figure 1: OpenFlow switch datapath

A packet enters the switch through a Port and is passed on the first Flow Table along with the Ingress Port as a Metadata (M1). Additionally each frame carries around a list of actions, called Action Set (M2), which have initially no actions in it. The Action Set will be executed at the end of the DataPath in the Execute Action Set block. After the first Flow Table another metadata called Metadata (M3) also accompanies the packet. This data inside the metadata maybe written by the Flow Tables when the Write Metadata instruction is applied.

Each Flow Table performs a match based on certain fields (e.g IP Source Address or Source MAC Address) and then perform a specific instruction if there is a match. If no match occurs, the frame is processed based on the Flow Table's configuration. The choices are either:

- Forward to the OpenFlow controller
- Send to the next flow table

c. Drop the frame

The list of instructions a Flow Table may perform upon a match are:

- o Apply a List of actions
- o Clear the Action Set
- o Write actions on the action set
- o Write Metadata
- o Go to Flow Table (allows a FlowTable X to send the packet and metadata to any FlowTable Y, provided that  $X > Y$ )

In OpenFlow there are two types of action executions which are independent of each other. The first one referred to as action list and is programmed into the flow table to be executed immediately within the packet pipeline upon a match on a flow table. The second one gets executed at the end of the pipeline in the execute action set. The second type of actions is collected in a metadata referred to as Action Set during the datapath processing with the Write Actions instruction.

The type of actions the Flow Table can perform or write in the Action Set is:

- o Setting of a field (e.g. IP address, MAC address)
- o Push or Pop tags (VLAN, MPLS)
- o Copy TTL inwards or outwards
- o Decrease TTLs
- o Output the packet to ports (a copy of the original packet will be sent to the port)
- o Apply QoS to a packet
- o Apply the packet to a group (a copy of the original packet will be sent to the group)

Additionally a Flow Table may drop the packet. The drop is implicit based on the Flow Table's configuration (e.g. when there are no more instructions).

An Action Set MUST contain a maximum of one action of each of the

following class of types which MUST be executed in the order specified below regardless of the order they were added to the Action Set. The output action in the action set is executed last. If both an output action and a group action are specified in an action set, the output action is ignored and the group action takes precedence. If no output action and no group action were specified in an action set, the packet is dropped.

1. Copy TTL inwards
2. Pop a tag (maximum one of VLAN tag, MPLS tag)
3. Push a tags (maximum one of VLAN tag, MPLS tag)
4. Copy TTL outwards
5. Decrease TTL
6. Setting of a field (maximum one of set IP address, set VLAN ID, etc...)
7. Apply QoS to a packet
8. Apply the packet to a group
9. Output the packet

The Group Table contains a set of Group Entries, each of which contains a set of Action Buckets, each of which contains a set of actions which can be applied to a group of packets that don't have the same set of matching fields. This alleviates the problem of having to set up the same set of actions in flow tables for different set of matching fields by having these set of actions in one place only.

### 3.2. ForCES-based OpenFlow Specification

ForCES models FEs using LFBs, fine-grained operations of the forwarding plane. It is logical to have at least the following LFB classes:

1. OFPort
2. OFFlowTables
3. OFGroupTable

#### 4. OFActions

#### 5. OFQueue

Additionally packets may be sent to the controller or the controller may send packets to the switch to be put on the datapath. RedirectIn and RedirectOut are two LFBs defined in the Base LFB Library [I-D.ietf-forces-lfb-lib]. However as some more metadata are required for the OpenFlow switch, the OFRedirectIn and OFRedirectOut will be defined by extending the initial LFBs.

While it may seem that having multiple OFFlowTables instances to represent each Flow Table in the OpenFlow, the authors decided to model the OFFlowTables to contain all the Flow Tables in one instance of the OFFlowTables. One of the OFFlowTables's components is an array of Flow Tables entries and each entry contains its own Flow Entries, Flow Table Counter and Miss Behaviour. The index of the Flow Tables entry represents the Flow Table ID. The rationale behind such a decision is the simplification of the model. With multiple ActionLFBs and multiple FlowTables, the resulting connection graph between Flow Tables and FlowTables and ActionLFBs would be very complex. Additionally this simplifies also the handling of two metadatas, the ActionSet Metadata and the Metadata which are now invisible to the model as they are passed only between Flow Tables. Figure 2 shows an example of how the OFFlowTables is internally.

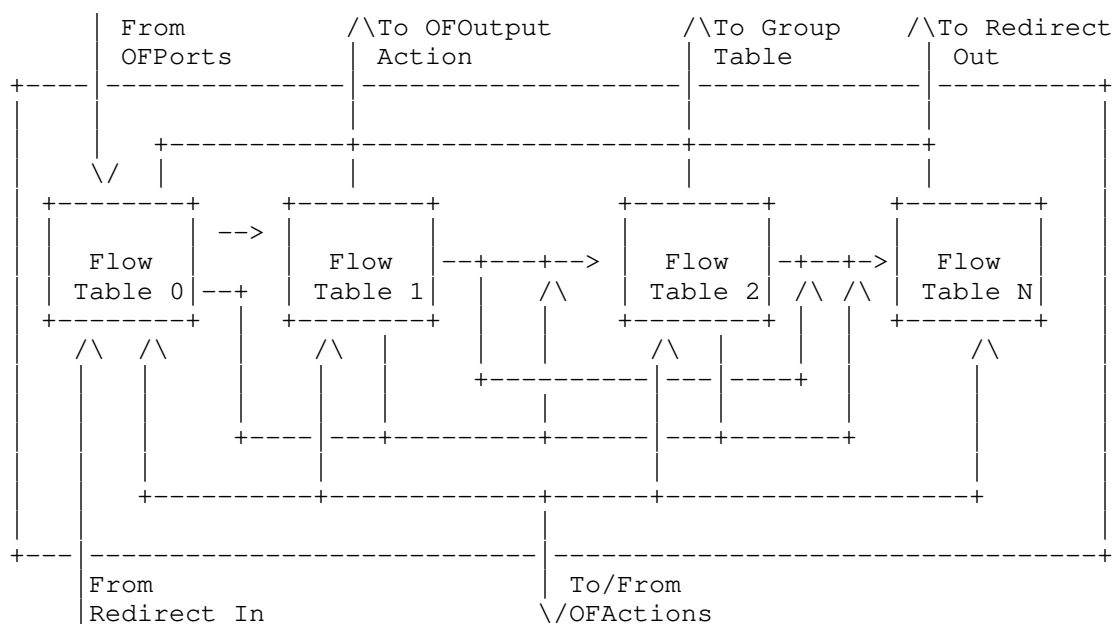


Figure 2: FlowTable Internal

Figure 3 depicts what an OFFlowTables with its own set of ActionLFBs would look like. Figure 4 depicts how the OFFlowTables work with a shared set of ActionLFBs with the OFGroupTable LFB look like.

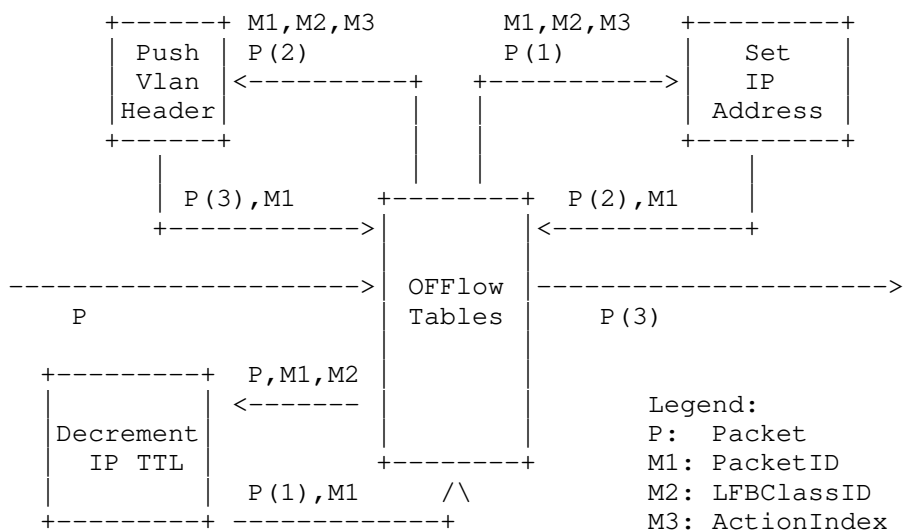


Figure 3: ForCES FlowTable with each own set of ActionLFBs

A packet P enters the OFFlowTables from an OFPortLFB. If a match occurs for the packet within the OFFlowTables and if the instruction for that match is an "Apply Action List" then the actions must be performed immediately and in the order specified in the action list. Alongside the packet, some metadata are passed as well. M1, the PacketID required by the OFFlowTables to identify the packet and continue execution from where it stopped when it returns, M2 the LFBClassID so that the ActionLFB knows the ClassID of the LFB to return the packet and M3 the ActionIndex required in some ActionLFBs to determine parameters for the action. For example in Figure 3 let's assume that the action list includes the following four actions:

1. Decrement IP TTL
2. Set IP Address
3. Push VLAN header
4. Final Action

The packet P will be first sent to the Decrement IP TTL Action LFB. Upon completion it will be returned as P1 to the OFFlowTables and then will be sent to the Set IP Address LFB. Upon return as P2 it will be sent to the Push VLAN Header and returned finally as P3 to the Flow Table LFB. Then depending upon the final action the packet may:

- o A copy of P3 will be sent to a Port LFB if it is an output action and the action set will be executed.
- o A copy of P3 will be sent to the Group LFB if it is an group action and the action set will be executed.
- o Remain in the OFFlowTables and checked for a match in the Flow Table specified by a Goto action if it is a goto action.

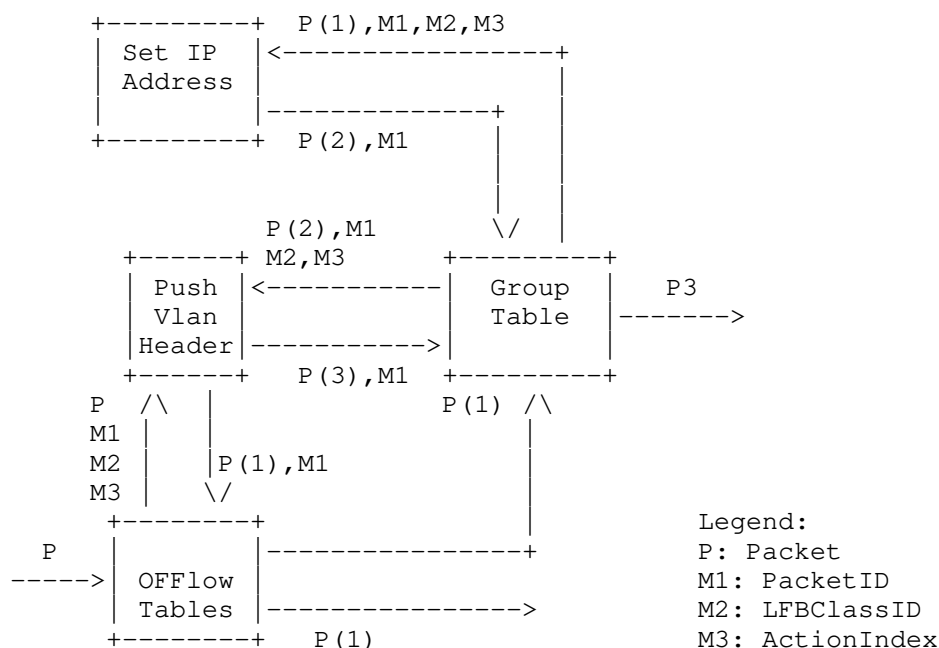


Figure 4: ForCES FlowTables and GroupTable with a common ActionLFB

A packet enters the OFFlowTables from an OFFortLFB. If a match occurs for the packet within the OFFlowTables and if the instruction for that match is an "Apply Action List" then the actions must be performed immediately and in the order specified in the action list. For example in Figure 4 let's assume that the action list includes the following three actions:

1. Push VLAN Header
2. Group
3. Output

The packet P will be first sent to the Push VLAN header Action LFB and upon completion will be returned as P1 to the OFFlowTables. Then a copy of P1 will be sent to the Group Table LFB and then finally a copy of P1 will be sent to the Port LFB for output and the action set will be executed.

Let's assume that the following actions will be executed for P1 in the action bucket in the Group Table.



1. Set IP Address
2. Push Vlan Header
3. Output

The packet P1 will be sent to the Set IP Address Action LFB and upon completion will be returned as P2 to the Group Table. Then the packet P2 will be sent to the Push Vlan Header Action LFB and be returned as P3 to the Group Table and be sent to the Port LFB for output.

Regarding the OFActions, when a match occurs in the OFFlowTables or an action bucket must be executed in the OFGroupTable it may contain multiple actions, it seems reasonable to separate each action as an individual LFB that performs that specific action. For every action needed to be executed, the OFFlowTables or the OFGroupTable will send the frame to the appropriate action LFB(s) in the order defined either by the Action Bucket or the instructions in the OFFlowTables's match entry. Once the packet has been processed from an Action LFB, it MUST be returned to the LFB instance that made that call. OFFlowTables and OFGroupTable LFB may either have shared ActionLFBs or separate.

Current specified Action LFBs are:

#### Output Actions

- o OFActionOutput

#### Set Queue Actions

- o OFActionSetQueue

#### Push/Pop Tag Actions

- o OFActionPushVlan
- o OFActionPopVLAN
- o OFActionPushMPLS
- o OFActionPopMPLS

#### Set Actions

- o OFActionSetMACSource
- o OFActionSetMACDestination
- o OFActionSetVLANVID
- o OFActionSetVLANPriority
- o OFActionSetMPLSLabel
- o OFActionSetMPLSTC
- o OFActionSetMPLSTTL
- o OFActionDecrementMPLSTTL
- o OFActionSetIPSource
- o OFActionSetIPDestination
- o OFActionSetIPTOS
- o OFActionSetIPECN
- o OFActionSetIPTTL
- o OFActionDecrementIPTTL
- o OFActionSetTCPSource
- o OFActionSetTCPDestination
- o OFActionCopyTTLOut
- o OFActionCopyTTLIn

#### Experimenter Actions

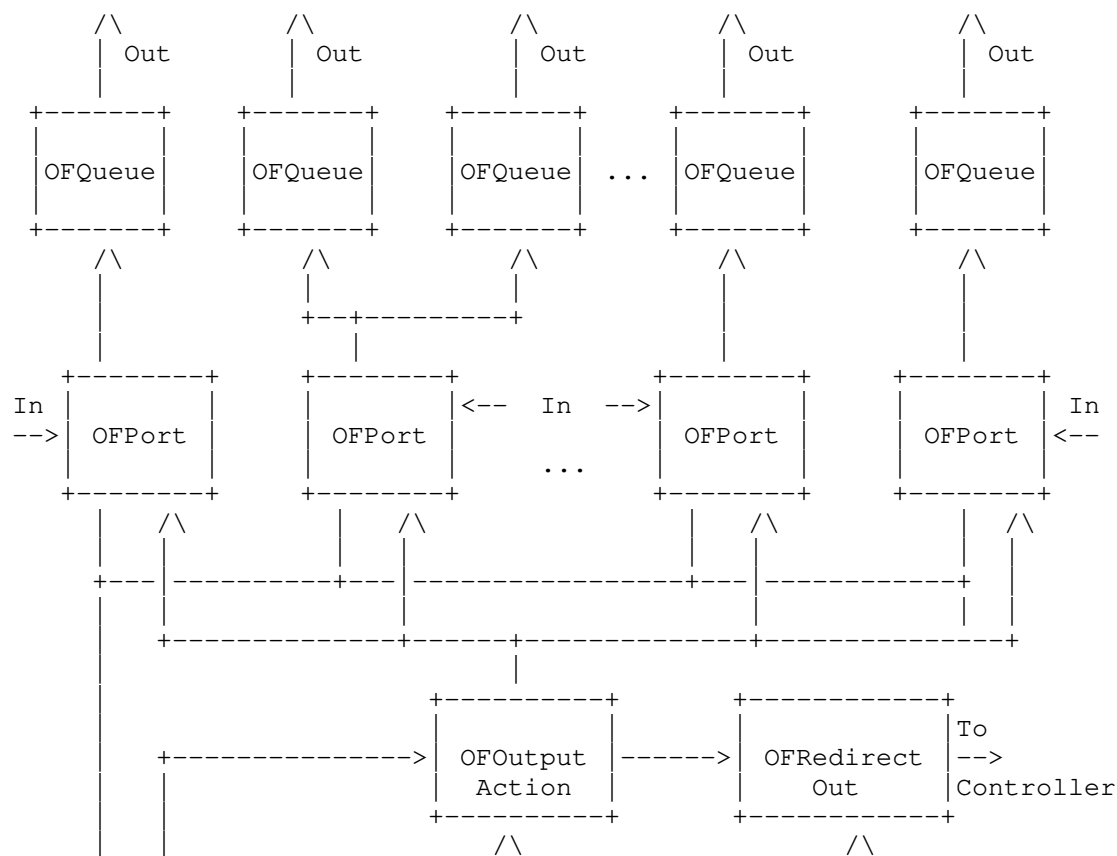
- o OFActionExperimenter

Most Action LFBs have data associated with the action, e.g. an IP Address for the SetIPSource or SetIPDestination actions, stored in an array in the LFB. The FlowTable sending the packet needs to send additionally as a metadata an index pointing to the action parameter needed for the execution. Each Action LFB has one group input port that accepts a packet, the LFBClassID of the LFB that sent the packet, so that it can be returned after the action has been performed, and optionally the Action Index. Furthermore, one more

metadata is required, the PacketID with which the Flow Table LFB or the Group Table LFB can keep track of a packet's progress. Additionally each Action LFB has one group output port that returns the altered packet to the sender. Since the action LFBs have these ports in common and the ForCES model can support augmentation of LFB classes, similar to inheritance in object oriented programming, an OFActionLFB has been specified from which all Action LFBs are derived from.

The Action LFBs can be used also by the OFGroupTable using the same input and output port.

Additionally each OFFlowTables can output a packet to a specific port through the OFOutputAction LFB. Figure 5 shows an example of a topology and how the various LFBs are interconnected. The controller can obtain the topology information by querying the FEObject's LFBTopology.



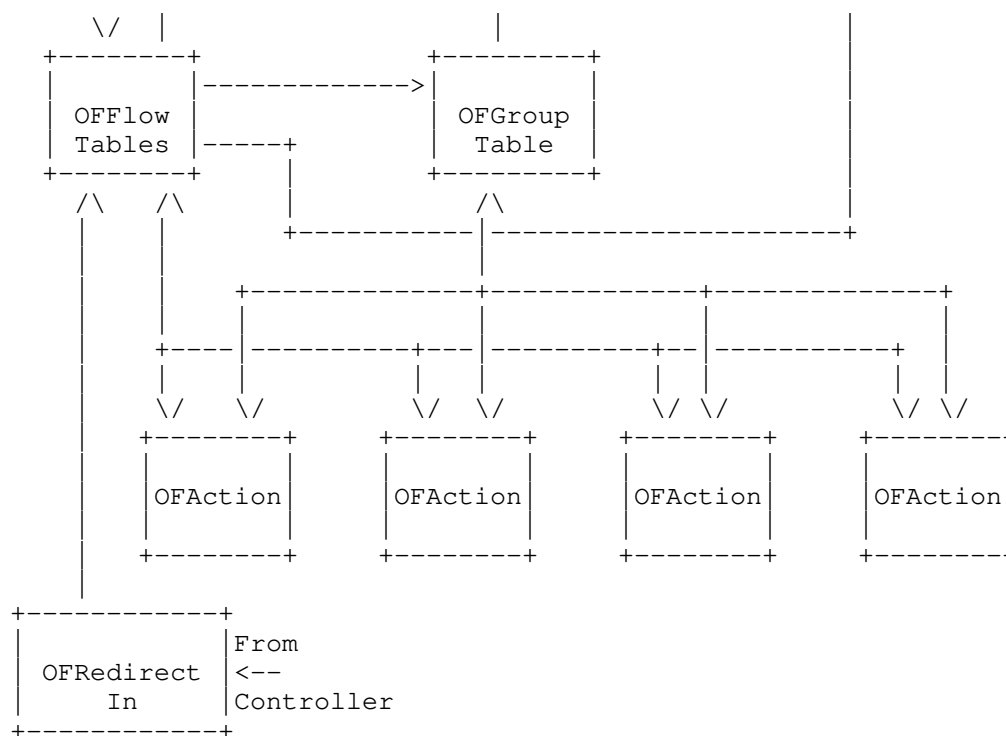


Figure 5: ForCES OpenFlow Switch example LFB connectivity

Regarding the execution of the Action Set, it is considered implementation specific and should be performed internally in the OFFlowTables using the ActionLFBs. Additionally the execution of OpenFlow's PacketOut message which contains a list of actions to be performed on a buffered or a redirected packet in the switch is also implementation specific and should also be performed internally in the OFFlowTables.

OpenFlow 1.1 provides, with the use of the experimenter concept new instruction or action types. In this model, new instruction types can be modeled by expanding the InstructionTypes datatype definition and new action types can be modeled by creating new ActionLFBs.

#### 4. OpenFlow Base Types

Some datatypes in this LFB library are imported from Base LFB Library [I-D.ietf-forces-lfb-lib] as they have already been defined there.

##### 4.1. Data Types

Data types defined in the OpenFlow library are categorized by types of atomic, compound struct, and compound array.

###### 4.1.1. Atomic

The following data types are defined as atomic data types in the OpenFlow library:

Data Type Name	Brief Description
MPLSLabelValue	An MPLS label
MPLSTrafficClassValues	The MPLS Traffic Class
IPv4ToSbits	TOSBits
ActionType	The possible actions
InstructionTypes	Instructions supported
FlowTableMissConfigType	Types to configure the default behavior of unmatched packets in a Flow Table
PacketInTypes	Packet In Types
GroupBucketExecuteType	To determine which Action Bucket(s) should be executed (all, select, indirect, fast failover)
PortNumberType	Port Number values
QueuePropertyType	Property type for a queue

OpenFlow Atomic Types

## 4.1.2. Compound Struct

The following data types are defined as struct data types in the OpenFlow library:

Data Type Name	Brief Description
SwitchDescriptionType	Fields of the switch description
WildcardsType	Wildcards for fields
MatchFieldType	A Match Field Type (contains all possible match fields)
FlowEntry	A Flow entry type
ActionRowType	An Action Row for the action table
TableCounterType	Counter per table
FlowCounterType	Counter per flow
WriteMetadataTableType	Metadata and mask for the write metadata instruction per row
GroupCounterType	Counters per group
BucketCounterType	Counters per bucket
GroupTableEntry	A Row of the Group Table
ActionBucket	An Action Bucket
PortConfigurationType	Types of configuration for the OpenFlow port
PortStateType	Current State of the port
PortFeaturesType	Port Features
PortCounterType	Counter per port
QueueArrayPropertiesType	Type Definition for property
QueueCounterType	Counters per queue

## OpenFlow Struct Types

## 4.1.3. Compound Array

The following data types are defined as an array data type in the OpenFlow library

Data Type Name	Brief Description
Actions	Actions to perform. An Array of ActionRowTypes

## OpenFlow Array Types

## 4.2. Frame Types

No additional frame types are defined in this library

## 4.3. MetaData Types

The following metadata are defined in the OpenFlow type library:

MetaData Name	Brief Description	MetaData ID	Metadata Type
IngressPort	The Ingress port the packet has arrived from.	1024	uint32
InPhyPort	The Port Index of the Physical interface the frame entered the switch	1025	uint32
PacketID	The PacketID metadata is used to uniquely identify a packet within the Flow Table or the Group Table to continue processing it after it has been returned from an OFActionLFB.	1026	uint32

ActionIndex	The Action Index metadata is used to point the row in the array in an Action LFB	1027	uint32
GroupIndex	The Group index metadata is used to point to the row of the array in an Group LFB	1028	uint32
LFBClassIDMetadata	The LFBClassID	1029	uint32
QueueIDMetadata	The Queue ID the packet should be sent to	1030	uint32
BufferID	The Buffer ID of a stored packet in the switch requiried for the PacketOut message	1031	uint32
RedirectReason	The reason the packet was redirected to the controller	1032	uchar
FlowTableID	The FlowTable ID the packet was sent to the controller from	1033	uchar
ActionListMetadata	The Action List that may come along with the packet in a PacketOut message	1032	octetstring

OpenFlow Metadata Types



## 5. OpenFlow LFBs

### 5.1. OpenFlowSwitch

Similar to the concept of the FEProtocol LFB and the FEObject LFB, the OpenFlowSwitchLFB contains information and configuration parameters regarding the functionality of the switch but play no role in the datapath processing. Therefore there are no input or output ports.

#### 5.1.1. Data Handling

This LFB does not handle any data.

#### 5.1.2. Components

The DatapathID component, a unsigned integer of 64 bits, uniquely identifies a datapath. The lower 48 bits are intended for the switch MAC address, while the top 16 bits are up to the implementer.

The MissSendLen component, an unsigned integer of 16 bits, defines the maximum number of bytes of each packet sent to the controller as a result of both flow table misses and flow table hits with the controller as the destination.

The HandleFragments component, a Boolean, defines what the switch does with fragments. If true the switch will drop fragments. If false there is no special handling.

The ReassembleFragments component, a Boolean, defines if the switch will reassemble fragments.

The InvalidTTLtoController component, a Boolean, defines whether the switch will send packets with invalid TTL to the controller.

The SwitchDescription component, a structure, contains the following information about the switch:

- o Manufacturer description
- o Hardware description
- o Software description
- o Serial Number
- o Human readable description of datapath

Lastly the Ports component is an array which contains in its rows, all the port numbers.

#### 5.1.3. Capabilities

The following capabilities have been defined for the OpenFlowSwitch LFB

An assortment of Boolean type capabilities to define:

- o FlowStatistics. If the switch keeps flow statistics
- o TableStatistics. If the switch keep table statistics
- o PortStatistics. If the switch keep port statistics
- o GroupStatistics. If the switch keep group statistics
- o IPReassembly. If the switch can reassemble IP fragments
- o QueueStats. If the switch keeps queue statistics
- o ARPMatchIP. If the switch matches IP addresses in ARP packets

The MaxBufferedPackets capability, an unsigned integer of 32 bits, defines the maximum packets the switch can buffer when sending packets to the controller.

The TablesSupported capability, an unsigned integer of 8 bits, defines the number of tables supported by the switch, each of which can have a different set of supported wildcard bits and number of entries.

Additionally the another capability, the ActionSupported, defines the supported actions for the switch.

#### 5.1.4. Events

Three events have been specified regarding the ports. The first event will be triggered when a new port is added to the switch, the second when a port has been removed from the switch and the third when a port has been modified

#### 5.2. OFFlowTables

An LFB that houses all OpenFlow Flow Tables residing in the switch.

### 5.2.1. Data Handling

The OFFlowTables describes the process of selecting packets and classify them into specific flows based on specific match fields assigned by the controller.

The LFB is expected to receive all types of Ethernet packets through a group input named InputPort from an OFPort along with the IngressPort and the InPhyPort as metadata.

All Flow Tables reside in an array within the LFB, the index of the array being the Flow Table ID.

Each Flow Table compares the packet with the MatchFields inside the FlowEntries Table. If there is no match, depending upon the MissBehaviour component, one of the following actions will occur:

1. The packet or part of it will be sent to the controller via the singleton output port RedirectPacketOut along with the IngressPort, InPhyPort, RedirectReason and FlowTableID as metadata and optionally the BufferID if the packet is buffered in the switch. How the packet is buffered in the switch is implementation specific.
2. The packet will be sent it to the next table in the pipeline (the next flow table row entry in the array).
3. The packet will be dropped.

If there is a match the LFB will decide based on the InstructionType of the component Instructions inside the matched FlowEntry.

If the instruction is Apply Actions, the LFB will use the InstructionIndex to find the Actions inside the ApplyActionList. Each row of the ApplyActionList is an array containing rows of ActionRowTypes. For every ActionRowType, the LFB will send the packet to the corresponding Action LFB through the group output ActionPort port alongside with the LFBClassIDMetadata of the LFB, the PacketID and the ActionIndex, if the specific action has any parameters, like the Set MAC Address action. The ActionIndex is used as an index for the table inside the Action LFB. The packet is then returned from the Action LFB through the group input port PacketReturn to continue further processing if exists. The PacketID is an identifier issued by the OFFlowTables LFB that will uniquely identify the packet so that when it returns it will continue processing from whence it stopped.

One exception to the rule when applying the action list is in regards

to the Group action. The OFGroup LFB handles groups. The OFFlowTables using the ActionIndex locates the Group Identifier in the OFFlowTables's GroupTable component. Then it sends a copy of the packet to the GroupTableLFB using the group output ActionPort and the original packet will continue in the Flow Table.

As the ActionSet metadata is an internal component of the OFFlowTables LFB, unreadable and unsettable by the Controller, both instructions Clear Actions and Write Actions are implementation specific. The same applies with the Write Metadata instruction.

If the instruction is Goto Table, the LFB will use the InstructionIndex to find the value OFFlowTables InstanceID in the GoToFlowTable table and send the packet to the FlowTables row entry with the index of the FlowTable ID.

Additionally the OFFlowTables handle packets incoming from the controller expecting either a whole packet through the RedirectPacketIn singleton input port or the buffer ID through the RedirectBufferIn singleton input port from the OFRedirectIn LFB, along with the ActionList and the IngressPort as metadata.

#### 5.2.2. Components

The FlowTables, an array holding all Flow Tables of the switch. Each row of the array is a struct of the FlowEntries, the FlowTableCounter and the MissBehaviour for each flow. The FlowEntries component of the struct is an array and each row of the array is a struct containing the cookie, the MatchFields, the Counters, the Instructions, the Timeouts, the Timers and the priority of the specific flow entry. The FlowTableCounter are the counters of the specific FlowTable and the MissBehaviour component of the struct specifies what the specific Flow Table shall do with the packet if there is no match.

The ApplyActionList is a component to maintain the actions required per flow. It is an array of Actions, which is an array of a struct of ActionType and ActionTableIndex.

The WriteActions is a component to maintain the actions to be written for a write actions instruction. It is an array of Actions, which is an array of a struct of ActionType and ActionTableIndex.

The WriteMetadataTable is a component to hold the metadata values required for the write metadata value. It is an array of WriteMetadataTableType, which is a struct of the Metadata value and the MetadataMask.

The GotoFlowTable component contains the FlowTable IDs flows should go to for the goto table action. It is an array of uint32. The value is selected using the InstructionIndex.

The GroupTable component contains group identifiers. It is an array of group identifiers indexed by the ActionTableIndex.

#### 5.2.3. Capabilities

This LFB has no capabilities specified.

#### 5.2.4. Events

One event has been defined regarding the Flow Table. The event will be triggered when a flow is deleted from the Flow Table whether due to the idle timeout, or to the hard timeout or a flow was deleted by the controller.

### 5.3. OFGroupTable

The Group LFB contains Action Buckets that can be applied to a packet during its path in the Flow Tables pipeline.

#### 5.3.1. Data Handling

The OFGroupTable does not take part in the actual handling of the data. Rather, it contains the action per group which are required by all Flow Tables in the pipeline. Packets initially enter this LFB from an OFActionSet LFB or a OFFlowTables via the group input port PacketIn and using the GroupIndex metadata the LFB finds the group requested for this packet. Then the LFB depending on the requested actions sends the packet to the required OFActionLFBs via the group output ActionPort and expects results via the group input PacketReturn.

#### 5.3.2. Components

The LFB has only one component which is the Group Table. This is an array of GroupTableEntry types. Each GroupTableEntry contains a Group Identifier, the type of Group, the required counters and an array of action buckets.

An action bucket is a struct which contains the Group weight required for select groups, the watch port and watch group required for fast failover groups, the bucket counters and the actions for this bucket.

The structure of actions in a bucket are identical to the actions in the flow table LFB containing the type of action and an action table

index. With the action type and action index the Group LFB can identify the component and index of the action details.

#### 5.3.3. Capabilities

This LFB has no capabilities specified.

#### 5.3.4. Events

This LFB has no events specified.

### 5.4. OFPort

#### 5.4.1. Data Handling

This LFB abstract the point where packets enter and exit the OpenFlow switch pipeline. May be a physical port, a virtual port defined by the switch. The LFB handles Ethernet frames coming in or going out to/of the OpenFlow switch. Ethernet frames are received and passed to an OFFlowTables through the singleton output port PacketIn, along with the IngressPortID and InPhyPort metadata.

When a packet is ready to be send on the wire, it is sent to an OFPort instance through the group input port PacketOut and then depending of whether the packet has been assigned a queue with the QueueID as metadata and then is sent to the OFQueue LFB via the group output port QueueOut.

#### 5.4.2. Components

The PortNumber component uniquely identifies the port within a switch.

The IEEEEMAC component contains the MAC Address of the port.

The Name component is a human readable name of the port.

The Configuration component specifies port behaviour. It's a struct component with the following boolean fields. PortDown, NoReceiving, NoForwarding and NoPacket\_In.

The State component defines the current state of the OpenFlow Switch. It is a struct component that defines whether the link is down, the port is blocked or the port can be used for live fast failover.

The Current Features component describes the current features of the port. It is a struct component and specifies the Speed Types, the Connected Medium, the Auto Negotiation and the Pause Types

The Advertised Features component describes the advertised features of the port. The component is of the same structure as the current features.

The CurrentSpeed component defines the current port bitrate in kbps.

The MaximumSpeed component defines the maximum port bitrate in kbps.

The PortCounter component contains the necessary counters for the port. It is a struct component comprised of counters for Packets Received, Packets Transmitted, Bytes Received, Bytes Transmitted, Drops Received, Transmit Drops, Errors in reception, Errors in transmission, Frame Alignment Errors received, Received Overrun Errors, CRC Errors in received packets, Collisions.

#### 5.4.3. Capabilities

Two capabilities has been defined for the Port LFB. Supported Features and Peer Features. These include:

- o Types of Speed supported
- o Medium Connected to the port
- o Auto-negotiation
- o Pause Types supported of the port

#### 5.4.4. Events

This LFB has no events specified.

### 5.5. OFQueue

#### 5.5.1. Data Handling

This LFB manages the queuing algorithm for handling packets prior to be output from the switch. Multiple OFQueue LFBs can be attached to an OFPort LFB to handle queues. If a packet has been set a QueueID with a Set Queue action, it is sent to the OFQueue LFB after the OFPort LFB and it enters this LFB via the group input port PacketIn where it will be handled according to the LFBs configuration and then be output from the switch.

#### 5.5.2. Components

The QueueID component, a uint32, defines the ID for the specific queue.

The Properties component, is an array of Property Types an the length of the property, defines the current queue mode. Current specified modes are none and minimum rate.

The QueueCounter component, a struct of TransmitPackets, TransmitBytes, TransimtOverrunErrors holds the necessary counter for the LFB.

#### 5.5.3. Capabilities

This LFB has no capabilities specified.

#### 5.5.4. Events

This LFB has no events specified.

### 5.6. OFRedirectIn

The OFRedirectIn LFB abstracts the process for the controller to inject data packets into the switch to input data packets into the data path. The LFB is derived from the RedirectIn LFB defined in the Base LFB Library [I-D.ietf-forces-lfb-lib].

#### 5.6.1. Data Handling

A packet or a bufferID arrives from the controller depending on whether the packet was buffered or not in the switch. If the packet was not buffered or the controller wishes to inject a packet into the switch, the packet will be output from the singleton output port PacketIn along with an ActionList and an IngressPort metadata to be sent to the OFFlowTables for processing. If the packet was buffered in the switch, the no packet is sent from the singleton output port BufferIn but only the BufferID, the ActionList and the IngressPort metadatas.

#### 5.6.2. Components

This LFB has no components specified.

#### 5.6.3. Capabilities

This LFB has no capabilities specified.

#### 5.6.4. Events

This LFB has no events specified.



### 5.7. OFRedirectOut

The OFRedirectOut LFB abstracts the process for the switch to deliver data packets to the controller. The LFB is derived from the RedirectIn LFB defined in the Base LFB Library [I-D.ietf-forces-lfb-lib].

#### 5.7.1. Data Handling

A packet or part of a packet along with the BufferID metadata arrives from the group input port. Outgoing from either the OFFlowTables or the OFActionOutput LFBs. Besides the optional BufferID metadata, the IngressPort, the InPhyPort and the RedirectReason and the optional FlowTableID are sent to the Outgoing input port required to be sent to the Controller as metadata.

#### 5.7.2. Components

This LFB has no components specified.

#### 5.7.3. Capabilities

This LFB has no capabilities specified.

#### 5.7.4. Events

This LFB has no events specified.

### 5.8. OFAction

This LFB is a template used for create OFActionLFBs. All OFActionLFBs have the input and output port in common but have different components. This LFB defines how input and output port of all OFActionLFBs. Inside OFActionLFBs there is a table with the required attributes where applicable as some OFActionLFBs don't require attributes.

#### 5.8.1. Data Handling

A packet arrives in an OFActionLFB via the group input PacketIn from an OFFlowTables or an OFGroupTable, along with the LFBClassID metadata, required to uniquely identify the sender, the PacketID and optionally the ActionIndex metadata if the action requires a specific attributes, the IngressPort, the InPhyPort and the QueueID required by the OFActionOutput. Once the packet has been processed it is return to the sender LFB via the group output PacketOut along with the PacketID.

#### 5.8.2. Components

This LFB has no components specified.

#### 5.8.3. Capabilities

This LFB has no capabilities specified.

#### 5.8.4. Events

This LFB has no events specified.

#### 5.9. OFActionLFBs

As none of the OFActionLFBs have any capabilities or events, these sections are omitted from the draft.

##### 5.9.1. OFActionOutput

###### 5.9.1.1. Data Handling

The OFActionOutputLFB does not modify the packet in any way, rather forwards a packet to a specified OFPort. Additionally there are several virtual ports that the OFActionOutputLFB may send the packet to:

All - Group output, sends the packet out all standard ports, but not to the ingress port or ports configured not to forward

Controller - Sends the packet to the controller

Table - Submit the packet to the first flow table so that the packet can be processed through the regular OpenFlow pipeline. Only valid in the action list of a packet-out message

InPort - Sends the packet out the ingress port.

Local - Sends the packet to the switch's local networking stack

Normal - Processes the packet using the traditional non-OpenFlow pipeline of the switch.

Flood - Floods the packet using the normal pipeline of the switch.

#### 5.9.1.2. Components

This LFB has only one component, the `OutputActionTable`, which is an array of a struct of the port number and optionally the maximum length in bytes, if the receiving end is the controller.

#### 5.9.2. OFActionSetVLANVID

##### 5.9.2.1. Data Handling

The `OFActionSetVLANVIDLFB` replaces the existing VLAN ID. Only applies to packets with an existing VLAN tag.

##### 5.9.2.2. Components

This LFB has only one component, the `SetVLANVIDActionTable`, which is an array of `uint16` VLAN tag values.

#### 5.9.3. OFActionSetVLANPriority

##### 5.9.3.1. Data Handling

The `OFActionSetVLANPriorityLFB` replaces the existing VLAN priority. Only applies to packets with an existing VLAN tag.

##### 5.9.3.2. Components

This LFB has only one component, the `SetVLANPriorityActionTable`, which is an array of `uchar` VLAN priority values.

#### 5.9.4. OFActionSetMACSource

##### 5.9.4.1. Data Handling

The `OFActionSetMACSourceLFB` replaces the existing Ethernet source MAC address.

##### 5.9.4.2. Components

This LFB has only one component, the `SetMACSourceActionTable`, which is an array of `IEEEMAC` addresses.

#### 5.9.5. OFActionSetMACDestination

##### 5.9.5.1. Data Handling

The `OFActionSetMACDestinationLFB` replaces the existing Ethernet source MAC address.

#### 5.9.5.2. Components

This LFB has only one component, the SetMACSourceActionTable, which is an array of IEEE MAC addresses.

#### 5.9.6. OFActionSetIPSource

##### 5.9.6.1. Data Handling

The OFActionSetIPSourceLFB replaces the existing IP source address with new value and update the IP checksum (and TCP/UDP/SCTP checksum if applicable). This action is only applicable to IPv4 packets.

##### 5.9.6.2. Components

This LFB has only one component, the SetIPSourceActionTable, which is an array of IPv4 addresses.

#### 5.9.7. OFActionSetIPDestination

##### 5.9.7.1. Data Handling

The OFActionSetIPDestinationLFB replaces the existing IP destination address with new value and update the IP checksum (and TCP/UDP/SCTP checksum if applicable). This action is only applicable to IPv4 packets.

##### 5.9.7.2. Components

This LFB has only one component, the SetIPDestinationActionTable, which is an array of IPv4 addresses.

#### 5.9.8. OFActionSetIPTOS

##### 5.9.8.1. Data Handling

The OFActionSetIPTOSLFB replaces the existing IP TOS value and update the IP checksum. Only applies to IPv4 packets.

##### 5.9.8.2. Components

This LFB has only one component, the SetIPTOSActionTable, which is an array of IPv4 uchar TOS values.

#### 5.9.9. OFActionSetIPECN

#### 5.9.9.1. Data Handling

The OFActionSetIPECNLFB replaces the existing IP ECN value and update the IP checksum. Only applies to IPv4 packets.

#### 5.9.9.2. Components

This LFB has only one component, the SetIPECNActionTable, which is an array of IPv4 uchar ECN values.

### 5.9.10. OFActionSetTCPSource

#### 5.9.10.1. Data Handling

The OFActionSetTCPSourceLFB replaces the existing TCP/UDP/SCTP source port with new value and update the TCP/UDP/SCTP checksum. This action is only applicable to TCP, UDP and SCTP packets.

#### 5.9.10.2. Components

This LFB has only one component, the SetTCPSourceActionTable, which is an array of uint16 values.

### 5.9.11. OFActionSetTCPDestination

#### 5.9.11.1. Data Handling

The OFActionSetTCPDestinationLFB replaces the existing TCP/UDP/SCTP destination port with new value and update the TCP/UDP/SCTP checksum. This action is only applicable to TCP, UDP and SCTP packets.

#### 5.9.11.2. Components

This LFB has only one component, the SetTCPDestinationActionTable, which is an array of uint16 values.

### 5.9.12. OFActionCopyTTLOut

#### 5.9.12.1. Data Handling

The OFActionCopyTTLOutLFB copies the TTL from next-to-outermost to outermost header with TTL. Copy can be IP-to-IP, MPLS-to-MPLS, or IP-to-MPLS.

#### 5.9.12.2. Components

This LFB has no components specified.

## 5.9.13. OFActionCopyTTLIn

## 5.9.13.1. Data Handling

The OFActionCopyTTLOutLFB copies the TTL from outermost to next-to-outermost header with TTL. Copy can be IP-to-IP, MPLS-to-MPLS, or IP-to-MPLS.

## 5.9.13.2. Components

This LFB has no components specified.

## 5.9.14. OFActionSetMPLSLabel

## 5.9.14.1. Data Handling

The OFActionSetMPLSLabelLFB replaces the existing MPLS label. Only applies to packets with an existing MPLS shim header.

## 5.9.14.2. Components

This LFB has only one component, the SetMPLSLabelActionTable, which is an array of uint32 MPLS label values.

## 5.9.15. OFActionSetMPLSTC

## 5.9.15.1. Data Handling

The OFActionSetMPLSTCLFB replaces the existing MPLS traffic class. Only applies to packets with an existing MPLS shim header.

## 5.9.15.2. Components

This LFB has only one component, the SetMPLSTCActionTable, which is an array of uchar MPLS label values.

## 5.9.16. OFActionSetMPLSTTL

## 5.9.16.1. Data Handling

The OFActionSetMPLSTTLLFB replaces the existing MPLS TTL. Only applies to packets with an existing MPLS shim header.

## 5.9.16.2. Components

This LFB has only one component, the SetMPLSTTLTable, which is an array of uchar MPLS TTL values.

## 5.9.17. OFActionDecrementMPLSTTL

## 5.9.17.1. Data Handling

The OFActionDecrementMPLSTTL LFB decrements the MPLS TTL. Only applies to packets with an existing MPLS shim header.

## 5.9.17.2. Components

This LFB has no components specified.

## 5.9.18. OFActionPushVlan

## 5.9.18.1. Data Handling

The OFActionPushVlan LFB pushes a new VLAN header onto the packet. The Ethertype is used as the Ethertype for the tag. Only Ethertype 0x8100 and 0x88a8 should be used.

## 5.9.18.2. Components

This LFB has only one component, the PushVLANTable, which is an array of uint16 ethertypes.

## 5.9.19. OFActionPopVLAN

## 5.9.19.1. Data Handling

The OFActionPopVLAN LFB pops the outer-most VLAN header from the packet.

## 5.9.19.2. Components

This LFB has no components specified.

## 5.9.20. OFPushMPLSOFAction

## 5.9.20.1. Data Handling

The OFPushMPLSOFAction LFB pushes a new MPLS shim header onto the packet. The Ethertype is used as the Ethertype for the tag. Only Ethertype 0x8847 and 0x8848 should be used.

## 5.9.20.2. Components

This LFB has only one component, the PushMPLSTable, which is an array of uint16 MPLS header values.

## 5.9.21. OFPopMPLSOFAction

## 5.9.21.1. Data Handling

The OFPopMPLSOFActionLFB pops the outer-most MPLS tag or shim header from the packet. The Ethertype is used as the Ethertype for the resulting packet (Ethertype for the MPLS payload).

## 5.9.21.2. Components

This LFB has only one component, the PopMPLSTable, which is an array of uint16 ethertype values.

## 5.9.22. OFSetQueueOFAction

## 5.9.22.1. Data Handling

The OFSetQueueOFActionLFB sets the queue ID for the packet. This LFB will return the packet along with the QueueID as a metadata.

## 5.9.22.2. Components

This LFB has only one component, the SetQueueTable, which is an array of uint32 queue identifiers.

## 5.9.23. OFSetIPTTLOFAction

## 5.9.23.1. Data Handling

The OFSetIPTTLOFActionLFB replaces the existing IP TTL and update the IP checksum. Only applies to IPv4 packets.

## 5.9.23.2. Components

This LFB has only one component, the SetIPTTLTable, which is an array of uchar TTL values.

## 5.9.24. OFDecrementIPTTLOFAction

## 5.9.24.1. Data Handling

The OFDecrementIPTTLOFActionLFB decrements the existing IP TTL and update the IP checksum. Only applies to IPv4 packets.

## 5.9.24.2. Components

This LFB has no components specified.



#### 5.9.25. OFExperimenterOFAction

##### 5.9.25.1. Data Handling

The OFExperimenterOFActionLFB handles experimenter actions.

##### 5.9.25.2. Components

This LFB has only one component, the SetIPTTLTable, which is an array of uint32 Experimenter ID values.

## 6. XML for OpenFlow library

```
<?xml version="1.0" encoding="UTF-8"?>
<LFBLibrary xmlns="urn:ietf:params:xml:ns:forces:lfbmodel:1.0"
  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  provides="OpenFlowLibrary">
  <load library="BaseTypeLibrary"
    location="urn:ietf:params:xml:ns:forces:lfbmodel:1.0"/>
  <dataTypeDefs>
    <!-- Data Type Definition for the OpenFlow Switch -->
    <dataTypeDef>
      <name>SwitchDescriptionType</name>
      <synopsis>The type of the switch description</synopsis>
      <struct>
        <component componentID="1">
          <name>MFR</name>
          <synopsis>Manufacturer description</synopsis>
          <typeRef>string[256]</typeRef>
        </component>
        <component componentID="2">
          <name>HW</name>
          <synopsis>Hardware description</synopsis>
          <typeRef>string[256]</typeRef>
        </component>
        <component componentID="3">
          <name>SF</name>
          <synopsis>Software description</synopsis>
          <typeRef>string[256]</typeRef>
        </component>
        <component componentID="4">
          <name>SerialNum</name>
          <synopsis>Serial Number</synopsis>
          <typeRef>string[32]</typeRef>
        </component>
        <component componentID="5">
          <name>DP</name>
          <synopsis>Human readable description of datapath</synopsis>
          <typeRef>string[256]</typeRef>
        </component>
      </struct>
    </dataTypeDef>
    <!-- Data Type Definition for the Flow Match -->
    <dataTypeDef>
      <name>MPLSLabelValue</name>
      <synopsis>An MPLS label.</synopsis>
      <atomic>
        <baseType>uint32</baseType>
```

```
<rangeRestriction>
  <allowedRange min="0" max="1048575"/>
</rangeRestriction>
</atomic>
</dataTypeDef>
<dataTypeDef>
  <name>MPLSTrafficClassValues</name>
  <synopsis>The MPLS Traffic Class</synopsis>
  <atomic>
    <baseType>uchar</baseType>
    <rangeRestriction>
      <allowedRange min="0" max="7"/>
    </rangeRestriction>
  </atomic>
</dataTypeDef>
<dataTypeDef>
  <name>IPv4ToSbits</name>
  <synopsis>TOSBits</synopsis>
  <atomic>
    <baseType>uchar</baseType>
    <rangeRestriction>
      <allowedRange min="0" max="63"/>
    </rangeRestriction>
  </atomic>
</dataTypeDef>
<dataTypeDef>
  <name>WildcardsType</name>
  <synopsis>Wildcards for fields</synopsis>
  <struct>
    <component componentID="1">
      <name>InPort</name>
      <synopsis>Input Port Wildcard</synopsis>
      <typeRef>boolean</typeRef>
    </component>
    <component componentID="2">
      <name>VLANID</name>
      <synopsis>Vlan ID Wildcard</synopsis>
      <typeRef>boolean</typeRef>
    </component>
    <component componentID="3">
      <name>VLANPCP</name>
      <synopsis>Vlan priority Wildcard</synopsis>
      <typeRef>boolean</typeRef>
    </component>
    <component componentID="4">
      <name>DLType</name>
      <synopsis>Ethernet frame typ Wildcard</synopsis>
      <typeRef>boolean</typeRef>
    </component>
  </struct>
</dataTypeDef>
```

```
</component>
<component componentID="5">
  <name>IPToS</name>
  <synopsis>IP ToS Wildcard</synopsis>
  <typeRef>boolean</typeRef>
</component>
<component componentID="6">
  <name>IPProtocol</name>
  <synopsis>IP Protocol Wildcard</synopsis>
  <typeRef>boolean</typeRef>
</component>
<component componentID="7">
  <name>TPSource</name>
  <synopsis>TCP/UDP/SCTP source port Wildcard</synopsis>
  <typeRef>boolean</typeRef>
</component>
<component componentID="8">
  <name>TPDestination</name>
  <synopsis>TCP/UDP/SCTP destination port Wildcard</synopsis>
  <typeRef>boolean</typeRef>
</component>
<component componentID="9">
  <name>MPLSLabel</name>
  <synopsis>MPLS label Wildcard</synopsis>
  <typeRef>boolean</typeRef>
</component>
<component componentID="10">
  <name>MPLSTC</name>
  <synopsis>MPLS TC Wildcard</synopsis>
  <typeRef>boolean</typeRef>
</component>
</struct>
</dataTypeDef>
<dataTypeDef>
  <name>MatchFieldType</name>
  <synopsis>A Match Field Type</synopsis>
  <struct>
    <component componentID="1">
      <name>IngressPort</name>
      <synopsis>Numerical representation of incoming port, starting
        at 1. This may be a physical or switch-defined virtual port.
      </synopsis>
      <typeRef>uint32</typeRef>
    </component>
    <component componentID="2">
      <name>Wildcards</name>
      <synopsis>Wildcards for match fields</synopsis>
      <typeRef>WildcardsType</typeRef>
    </component>
  </struct>
</dataTypeDef>
```

```
</component>
<component componentID="3">
  <name>EthernetSourceAddress</name>
  <synopsis>Ethernet source address</synopsis>
  <typeRef>IEEEEMAC</typeRef>
</component>
<component componentID="4">
  <name>EthernetSourceAddressMask</name>
  <synopsis>Ethernet source address mask</synopsis>
  <typeRef>IEEEEMAC</typeRef>
</component>
<component componentID="5">
  <name>EthernetDestinationAddress</name>
  <synopsis>Ethernet destination address</synopsis>
  <typeRef>IEEEEMAC</typeRef>
</component>
<component componentID="6">
  <name>EthernetDestinationAddressMask</name>
  <synopsis>Ethernet destination address mask</synopsis>
  <typeRef>IEEEEMAC</typeRef>
</component>
<component componentID="7">
  <name>VlanID</name>
  <synopsis>VLAN identifier of outermost VLAN tag.</synopsis>
  <typeRef>VlanIDType</typeRef>
</component>
<component componentID="8">
  <name>VlanPriority</name>
  <synopsis>VLAN PCP Field of outermost VLAN tag.</synopsis>
  <typeRef>VlanPriorityType</typeRef>
</component>
<component componentID="9">
  <name>EtherType</name>
  <synopsis>Ethernet type of the OpenFlow packet payload, after
  VLAN tags. 802.3 frames have special handling.</synopsis>
  <typeRef>uint16</typeRef>
</component>
<component componentID="10">
  <name>IPv4TOS</name>
  <synopsis>Specify as 8-bit value and place ToS in upper 6
  bits for match</synopsis>
  <typeRef>IPv4ToSbits</typeRef>
</component>
<component componentID="11">
  <name>IPProto</name>
  <synopsis>IP protocol or lower 8 bits of
  ARP opcode. Only the lower 8 bits of the ARP opcode are used
  for the match</synopsis>
```

```
<typeRef>uchar8</typeRef>
</component>
<component componentID="12">
  <name>IPv4SourceAddress</name>
  <synopsis>IPv4 Source Address to match</synopsis>
  <typeRef>IPv4Addr</typeRef>
</component>
<component componentID="13">
  <name>IPv4SourceAddressMask</name>
  <synopsis>IPv4 Source Address mask</synopsis>
  <typeRef>IPv4Addr</typeRef>
</component>
<component componentID="14">
  <name>IPv4DestinationAddress</name>
  <synopsis>IPv4 Destination Address to match</synopsis>
  <typeRef>IPv4Addr</typeRef>
</component>
<component componentID="15">
  <name>IPv4DestinationAddressMask</name>
  <synopsis>IPv4 Destination Address mask</synopsis>
  <typeRef>IPv4Addr</typeRef>
</component>
<component componentID="16">
  <name>TCPSourcePort</name>
  <synopsis>Source Port for TCP and ICMP to match</synopsis>
  <typeRef>uint16</typeRef>
</component>
<component componentID="17">
  <name>TCPDestinationPort</name>
  <synopsis>Destination Port for TCP and ICMP to match
  </synopsis>
  <typeRef>uint16</typeRef>
</component>
<component componentID="18">
  <name>MPLSLabel</name>
  <synopsis>Match on outermost MPLS tag.</synopsis>
  <typeRef>MPLSLabelValue</typeRef>
</component>
<component componentID="19">
  <name>MPLSTrafficClass</name>
  <synopsis>Match on outermost MPLS tag for traffic class.
  </synopsis>
  <typeRef>MPLSTrafficClassValues</typeRef>
</component>
<component componentID="20">
  <name>Metadata</name>
  <synopsis>Meta Data</synopsis>
  <typeRef>uint64</typeRef>
```

```

    </component>
    <component componentID="21">
      <name>MetadataMask</name>
      <synopsis>Meta Data Mask</synopsis>
      <typeRef>uint64</typeRef>
    </component>
  </struct>
</dataTypeDef>
<!-- Datatype Definition for Flow Table -->
<dataTypeDef>
  <name>FlowEntry</name>
  <synopsis>A Flow entry</synopsis>
  <struct>
    <component componentID="1">
      <name>Cookie</name>
      <synopsis>Opaque data chosen by controller</synopsis>
      <typeRef>uint64</typeRef>
    </component>
    <component componentID="2">
      <name>MatchFields</name>
      <synopsis>Match Fields: to match against packets. These
        consist of the ingress port and packet headers, and
        optionally metadata specified by a previous table</synopsis>
      <typeRef>MatchFieldType</typeRef>
    </component>
    <component componentID="3">
      <name>Counters</name>
      <synopsis>Counters: to update for matching packets</synopsis>
      <typeRef>FlowCounterType</typeRef>
    </component>
    <component componentID="4">
      <name>Instructions</name>
      <synopsis>Instruction: what to do with the packet of the flow
    </synopsis>
    <array>
      <struct>
        <component componentID="1">
          <name>InstructionType</name>
          <synopsis>The instruction type</synopsis>
          <typeRef>InstructionTypes</typeRef>
        </component>
        <component componentID="2">
          <name>InstructionIndex</name>
          <synopsis>The instruction index.</synopsis>
          <typeRef>uint32</typeRef>
        </component>
      </struct>
    <contentKey contentKeyID="1">

```

```

        <contentKeyField>InstructionType</contentKeyField>
    </contentKey>
</array>
</component>
<component componentID="5">
    <name>Timeouts</name>
    <synopsis>Timeouts for the flow entry</synopsis>
    <struct>
        <component componentID="1">
            <name>IdleTimeout</name>
            <synopsis>Timeout to expire if no flows are matched for
            this flow entry</synopsis>
            <typeRef>uint16</typeRef>
        </component>
        <component componentID="2">
            <name>HardTimeout</name>
            <synopsis>Timeout to expire for this flow entry
            regardless of idle timeout</synopsis>
            <typeRef>uint16</typeRef>
        </component>
    </struct>
</component>
<component componentID="6">
    <name>Timers</name>
    <synopsis>Timers per flow</synopsis>
    <struct>
        <component componentID="1">
            <name>Duration_Sec</name>
            <synopsis>Time flow has been alive in seconds</synopsis>
            <typeRef>uint32</typeRef>
        </component>
        <component componentID="2">
            <name>Duration_nSec</name>
            <synopsis>Time flow has been alive in nanoseconds beyond
            Duration_Sec</synopsis>
            <typeRef>uint32</typeRef>
        </component>
    </struct>
</component>
<component componentID="7">
    <name>Priority</name>
    <synopsis>Priority within the specified flow table</synopsis>
    <typeRef>uint16</typeRef>
</component>
</struct>
</dataTypeDef>
<dataTypeDef>
    <name>ActionRowType</name>

```



```

<synopsis>An Action Row for the action table</synopsis>
<struct>
  <component componentID="1">
    <name>Action</name>
    <synopsis>The type of action</synopsis>
    <typeRef>ActionType</typeRef>
  </component>
  <component componentID="2">
    <name>ActionTableIndex</name>
    <synopsis>Index of the Table this action applies to
    </synopsis>
    <typeRef>uint32</typeRef>
  </component>
</struct>
</dataTypeDef>
<dataTypeDef>
  <name>ActionType</name>
  <synopsis>The type of action</synopsis>
  <atomic>
    <baseType>uint16</baseType>
    <specialValues>
      <specialValue value="1">
        <name>OUTPUT</name>
        <synopsis>Output to switch port</synopsis>
      </specialValue>
      <specialValue value="2">
        <name>SetVLANVID</name>
        <synopsis>Set the 802.1q VLAN id</synopsis>
      </specialValue>
      <specialValue value="3">
        <name>SetVLANPCP</name>
        <synopsis>Set the 802.1q priority</synopsis>
      </specialValue>
      <specialValue value="4">
        <name>SetDLSrc</name>
        <synopsis>Set Ethernet source address</synopsis>
      </specialValue>
      <specialValue value="5">
        <name>SetDLDst</name>
        <synopsis>Set Ethernet destination address</synopsis>
      </specialValue>
      <specialValue value="6">
        <name>SetIPSrc</name>
        <synopsis>Set IP source address</synopsis>
      </specialValue>
      <specialValue value="7">
        <name>SetIPDst</name>
        <synopsis>Set IP Destination address</synopsis>
      </specialValue>
    </specialValues>
  </atomic>
</dataTypeDef>

```

```
</specialValue>
<specialValue value="8">
  <name>SetIPTOS</name>
  <synopsis>Set ToS field</synopsis>
</specialValue>
<specialValue value="9">
  <name>SetIPECN</name>
  <synopsis>Set ECN field</synopsis>
</specialValue>
<specialValue value="10">
  <name>SetTPSource</name>
  <synopsis>TCP/UDP/SCTP source port</synopsis>
</specialValue>
<specialValue value="11">
  <name>SetTPDestination</name>
  <synopsis>TCP/UDP/SCTP destination port</synopsis>
</specialValue>
<specialValue value="12">
  <name>CopyTTLOut</name>
  <synopsis>Copy TTL "outwards" -- from next-to-outermost to
    outermost</synopsis>
</specialValue>
<specialValue value="13">
  <name>CopyTTLIn</name>
  <synopsis>Copy TTL "inwards" -- from outermost to
    next-to-outermost</synopsis>
</specialValue>
<specialValue value="14">
  <name>SetMPLSLabel</name>
  <synopsis>Set MPLS label</synopsis>
</specialValue>
<specialValue value="15">
  <name>SetMPLSTC</name>
  <synopsis>Set MPLS TC</synopsis>
</specialValue>
<specialValue value="16">
  <name>SetMPLSTTL</name>
  <synopsis>Set MPLS TTL</synopsis>
</specialValue>
<specialValue value="17">
  <name>PushVLANTag</name>
  <synopsis>Push a new VLAN tag</synopsis>
</specialValue>
<specialValue value="18">
  <name>PopVLANTag</name>
  <synopsis>Pop the outer VLAN tag</synopsis>
</specialValue>
<specialValue value="19">
```

```
        <name>PushMPLSTag</name>
        <synopsis>Push a new MPLS tag</synopsis>
    </specialValue>
    <specialValue value="20">
        <name>PopMPLSTag</name>
        <synopsis>Pop the outer MPLS tag</synopsis>
    </specialValue>
    <specialValue value="21">
        <name>SetQueue</name>
        <synopsis>Set queue ID when outputting to a port</synopsis>
    </specialValue>
    <specialValue value="22">
        <name>Group</name>
        <synopsis>Apply group</synopsis>
    </specialValue>
    <specialValue value="23">
        <name>SetIPTTL</name>
        <synopsis>Set IP TTL</synopsis>
    </specialValue>
    <specialValue value="24">
        <name>DecIPTTL</name>
        <synopsis>Decrement IP TTL</synopsis>
    </specialValue>
</specialValues>
</atomic>
</dataTypeDef>
<dataTypeDef>
    <name>TableCounterType</name>
    <synopsis>Counter per table</synopsis>
    <struct>
        <component componentID="1">
            <name>ReferenceCount</name>
            <synopsis>Active Entries</synopsis>
            <typeRef>uint32</typeRef>
        </component>
        <component componentID="2">
            <name>PacketLookups</name>
            <synopsis>Packet Lookups</synopsis>
            <typeRef>uint64</typeRef>
        </component>
        <component componentID="3">
            <name>PacketMatches</name>
            <synopsis>Packet Matches</synopsis>
            <typeRef>uint64</typeRef>
        </component>
    </struct>
</dataTypeDef>
<dataTypeDef>
```

```

    <name>Actions</name>
    <synopsis>Actions to perform. An Array of ActionRowTypes
    </synopsis>
    <array>
      <typeRef>ActionRowType</typeRef>
    </array>
  </dataTypeDef>
  <dataTypeDef>
    <name>FlowCounterType</name>
    <synopsis>Counter per flow</synopsis>
    <struct>
      <component componentID="1">
        <name>ReceivedPackets</name>
        <synopsis>Packets Received</synopsis>
        <typeRef>uint64</typeRef>
      </component>
      <component componentID="2">
        <name>ReceivedBytes</name>
        <synopsis>Bytes Received</synopsis>
        <typeRef>uint64</typeRef>
      </component>
      <component componentID="3">
        <name>DurationS</name>
        <synopsis>Duration in seconds</synopsis>
        <typeRef>uint32</typeRef>
      </component>
      <component componentID="4">
        <name>DurationNS</name>
        <synopsis>Duration in nanoseconds</synopsis>
        <typeRef>uint32</typeRef>
      </component>
    </struct>
  </dataTypeDef>
  <dataTypeDef>
    <name>InstructionTypes</name>
    <synopsis>Instructions supported</synopsis>
    <atomic>
      <baseType>short</baseType>
      <specialValues>
        <specialValue value="1">
          <name>GotoTable</name>
          <synopsis>Indicates the next table in the processing
            pipeline. The table-id must be greater than the current
            table-id. The flows of last table of the pipeline can not
            include this instruction</synopsis>
        </specialValue>
        <specialValue value="2">
          <name>WriteMetadata</name>

```

```

        <synopsis>Writes the masked metadata value into the
        metadata field. The mask specifies which bits of the
        metadata register should be modified (i.e. new metadata =
        old metadata and ~mask | value and mask)</synopsis>
    </specialValue>
    <specialValue value="3">
        <name>WriteAction</name>
        <synopsis>Merges the specified action(s) into the current
        action set. If an action of the given type exists in the
        current set, overwrite it, otherwise add it.</synopsis>
    </specialValue>
    <specialValue value="4">
        <name>ApplyActions</name>
        <synopsis>Applies the specific action(s) immediately,
        without any change to the Action Set. This instruction may
        be used to modify the packet between two tables or to
        execute multiple actions of the same type. The actions are
        specified as an action list</synopsis>
    </specialValue>
    <specialValue value="5">
        <name>ClearActions</name>
        <synopsis>Clears all the actions in the action set
        immediately.</synopsis>
    </specialValue>
    </specialValues>
</atomic>
</dataTypeDef>
<dataTypeDef>
    <name>WriteMetadataTableType</name>
    <synopsis>Metadata and mask for the write metadata instruction
    per row</synopsis>
    <struct>
        <component componentID="1">
            <name>Metadata</name>
            <synopsis>The metadata</synopsis>
            <typeRef>uint64</typeRef>
        </component>
        <component componentID="2">
            <name>MetadataMask</name>
            <synopsis>The metadata mask</synopsis>
            <typeRef>uint64</typeRef>
        </component>
    </struct>
</dataTypeDef>
<dataTypeDef>
    <name>FlowTableMissConfigType</name>
    <synopsis>Types to configure the default behavior of unmatched
    packets in a Flow Table</synopsis>

```

```

    <atomic>
      <baseType>uint32</baseType>
      <specialValues>
        <specialValue value="0">
          <name>Controller</name>
          <synopsis>Send to the controller</synopsis>
        </specialValue>
        <specialValue value="1">
          <name>Continue</name>
          <synopsis>Continue to the next table in the pipeline or
            send to the controller if the FlowTable is the last.
          </synopsis>
        </specialValue>
        <specialValue value="2">
          <name>Drop</name>
          <synopsis>Drop the packet</synopsis>
        </specialValue>
      </specialValues>
    </atomic>
  </dataTypeDef>
  <dataTypeDef>
    <name>PacketInTypes</name>
    <synopsis>Packet-in Types</synopsis>
    <atomic>
      <baseType>uchar</baseType>
      <specialValues>
        <specialValue value="0">
          <name>NoMatch</name>
          <synopsis>No Matching flow</synopsis>
        </specialValue>
        <specialValue value="1">
          <name>Action</name>
          <synopsis>Explicit action to send to controller</synopsis>
        </specialValue>
      </specialValues>
    </atomic>
  </dataTypeDef>
  <!-- Data Type Definition for the group -->
  <dataTypeDef>
    <name>GroupCounterType</name>
    <synopsis>Counters per group</synopsis>
    <struct>
      <component componentID="1">
        <name>ReferenceCount</name>
        <synopsis>Flow Entries</synopsis>
        <typeRef>uint32</typeRef>
      </component>
      <component componentID="2">

```

```

        <name>PacketCount</name>
        <synopsis>Packet Count</synopsis>
        <typeRef>uint64</typeRef>
    </component>
    <component componentID="3">
        <name>ByteCount</name>
        <synopsis>Byte Count</synopsis>
        <typeRef>uint64</typeRef>
    </component>
</struct>
</dataTypeDef>
<dataTypeDef>
    <name>GroupBucketExecuteType</name>
    <synopsis>To determine which Action Bucket(s) should be
        executed</synopsis>
    <atomic>
        <baseType>uchar</baseType>
        <specialValues>
            <specialValue value="1">
                <name>all</name>
                <synopsis>Execute all buckets in the group. This group is
                    used for multicast or broadcast forwarding. The packet is
                    effectively cloned for each bucket; one packet is processed
                    for each bucket of the group. If a bucket directs a packet
                    explicitly out the ingress port, this packet clone is
                    dropped. If the controller writer wants to forward out the
                    ingress port, the group should include an extra bucket
                    which includes an output action to the OFPP_IN_PORT virtual
                    port.</synopsis>
            </specialValue>
            <specialValue value="2">
                <name>select</name>
                <synopsis>Execute one bucket in the group. Packets are sent
                    to a single bucket in the group, based on a switch-computed
                    selection algorithm (e.g. hash on some user-configured
                    tuple or simple round robin). All configuration and state
                    for the selection algorithm is external to OpenFlow. When a
                    port specified in a bucket in a select group goes down,
                    the switch may restrict bucket selection to the remaining
                    set (those with forwarding actions to live ports) instead
                    of dropping packets destined to that port. This behavior
                    may reduce the disruption of a downed link or switch.
                </synopsis>
            </specialValue>
            <specialValue value="3">
                <name>indirect</name>
                <synopsis>Execute the one defined bucket in this group.
                    Allows multiple flows or groups to point to a common group

```

```

        identifier, supporting faster, more efficient convergence
        (e.g. next hops for IP forwarding). This group type is
        effectively identical to an all group with one bucket.
    </synopsis>
</specialValue>
<specialValue value="4">
    <name>fastfailover</name>
    <synopsis>Execute the first live bucket. Each action bucket
    is associated with a specific port and/or group that
    controls its liveness. Enables the switch to change
    forwarding without requiring a round trip to the
    controller. If no buckets are live, packets are dropped.
    This group type must implement a liveness
    mechanism.</synopsis>
</specialValue>
</specialValues>
</atomic>
</dataTypeDef>
<dataTypeDef>
    <name>BucketCounterType</name>
    <synopsis>Counters per bucket</synopsis>
    <struct>
        <component componentID="1">
            <name>PacketCount</name>
            <synopsis>Packet Count</synopsis>
            <typeRef>uint64</typeRef>
        </component>
        <component componentID="2">
            <name>ByteCount</name>
            <synopsis>Byte Count</synopsis>
            <typeRef>uint64</typeRef>
        </component>
    </struct>
</dataTypeDef>
<dataTypeDef>
    <name>GroupTableEntry</name>
    <synopsis>A Row of the Group Table</synopsis>
    <struct>
        <component componentID="1">
            <name>GroupID</name>
            <synopsis>Group Identifier uniquely identifying the group
            </synopsis>
            <typeRef>uint32</typeRef>
        </component>
        <component componentID="2">
            <name>GroupType</name>
            <synopsis>The group type to determine which action bucket
            will be executed.</synopsis>

```



```
        <typeRef>GroupBucketExecuteType</typeRef>
    </component>
    <component componentID="3">
        <name>GroupCounters</name>
        <synopsis>Counters per group</synopsis>
        <typeRef>GroupCounterType</typeRef>
    </component>
    <component componentID="4">
        <name>ActionBuckets</name>
        <synopsis>An ordered list of action buckets. Each action
        bucket is a set of actions similar to a flow table</synopsis>
        <array>
            <typeRef>ActionBucket</typeRef>
        </array>
    </component>
</struct>
</dataTypeDef>
<dataTypeDef>
    <name>ActionBucket</name>
    <synopsis>An Action Bucket</synopsis>
    <struct>
        <component componentID="1">
            <name>Weight</name>
            <synopsis>Relative weight of bucket. Only defined for select
            groups.</synopsis>
            <typeRef>uint16</typeRef>
        </component>
        <component componentID="2">
            <name>WatchPort</name>
            <synopsis>Port whose state affects whether this bucket is
            live. Required for fast failover group</synopsis>
            <typeRef>uint32</typeRef>
        </component>
        <component componentID="3">
            <name>WatchGroup</name>
            <synopsis>Group whose state affects whether this group is
            live. Only required for fast failover groups</synopsis>
            <typeRef>uint32</typeRef>
        </component>
        <component componentID="4">
            <name>Actions</name>
            <synopsis>Actions for this bucket</synopsis>
            <typeRef>Actions</typeRef>
        </component>
        <component componentID="5">
            <name>BucketCounter</name>
            <synopsis>A counter for this bucket</synopsis>
            <typeRef>BucketCounterType</typeRef>
```

```
        </component>
    </struct>
</dataTypeDef>
<!-- Data Type Definition for ports -->
<dataTypeDef>
    <name>PortNumberType</name>
    <synopsis>Port Number values</synopsis>
    <atomic>
        <baseType>uint32</baseType>
        <specialValues>
            <specialValue value="0xffffffff8">
                <name>InPort</name>
                <synopsis>Sent the packet out the input port. This virtual
                    port must be explicitly used in order to send back out of
                    the input port</synopsis>
            </specialValue>
            <specialValue value="0xffffffff9">
                <name>Table</name>
                <synopsis>Submit the packet to the first flow table. NBL
                    This destination port can only be used in packet-out
                    messages</synopsis>
            </specialValue>
            <specialValue value="0xffffffffffa">
                <name>Normal</name>
                <synopsis>Process with normal L2/L3 switching</synopsis>
            </specialValue>
            <specialValue value="0xffffffffffb">
                <name>Flood</name>
                <synopsis>Send the packet to all physical ports in VLAN,
                    except input port and those blocked or link down</synopsis>
            </specialValue>
            <specialValue value="0xffffffffffc">
                <name>All</name>
                <synopsis>Send the packet to all physical ports, except
                    input port.</synopsis>
            </specialValue>
            <specialValue value="0xffffffffffd">
                <name>Controller</name>
                <synopsis>Send the packet to the controller.</synopsis>
            </specialValue>
            <specialValue value="0xffffffffffe">
                <name>Local</name>
                <synopsis>Local openflow "port".</synopsis>
            </specialValue>
            <specialValue value="0xfffffffffff">
                <name>Any</name>
                <synopsis>Wildcard port used only for flow mod (delete) and
                    flow stats requests. Selects all flows regardless of output
```

```
        port (including flows with no output port).</synopsis>
    </specialValue>
</specialValues>
</atomic>
</dataTypeDef>
<dataTypeDef>
  <name>PortConfigurationType</name>
  <synopsis>Types of configuration for the OpenFlow port</synopsis>
  <struct>
    <component componentID="1">
      <name>PortDown</name>
      <synopsis>Port is administratively down</synopsis>
      <typeRef>boolean</typeRef>
    </component>
    <component componentID="2">
      <name>NoReceiving</name>
      <synopsis>Drop all packets received by this port</synopsis>
      <typeRef>boolean</typeRef>
    </component>
    <component componentID="3">
      <name>NoForwarding</name>
      <synopsis>Drop packets forwarded to the port</synopsis>
      <typeRef>boolean</typeRef>
    </component>
    <component componentID="4">
      <name>NoPacket_In</name>
      <synopsis>Do not send packet-in messages for port</synopsis>
      <typeRef>boolean</typeRef>
    </component>
  </struct>
</dataTypeDef>
<dataTypeDef>
  <name>PortStateType</name>
  <synopsis>Current State of the port</synopsis>
  <struct>
    <component componentID="1">
      <name>LinkDown</name>
      <synopsis>No physical link present</synopsis>
      <typeRef>boolean</typeRef>
    </component>
    <component componentID="2">
      <name>PortBlocked</name>
      <synopsis>Port is blocked</synopsis>
      <typeRef>boolean</typeRef>
    </component>
    <component componentID="3">
      <name>PortLive</name>
      <synopsis>Live for Fast Failover Group</synopsis>
```

```
        <typeRef>boolean</typeRef>
      </component>
    </struct>
  </dataTypeDef>
  <dataTypeDef>
    <name>PortFeaturesType</name>
    <synopsis>Port Features</synopsis>
    <struct>
      <component componentID="1">
        <name>SpeedTypes</name>
        <synopsis>Types of Speed supported</synopsis>
        <struct>
          <component componentID="1">
            <name>10MB_HD</name>
            <synopsis>10 Mb half-duplex rate support.</synopsis>
            <typeRef>boolean</typeRef>
          </component>
          <component componentID="2">
            <name>10MB_FD</name>
            <synopsis>10 Mb full-duplex rate support.</synopsis>
            <typeRef>boolean</typeRef>
          </component>
          <component componentID="3">
            <name>100MB_HD</name>
            <synopsis>100 Mb half-duplex rate support.</synopsis>
            <typeRef>boolean</typeRef>
          </component>
          <component componentID="4">
            <name>100MB_FD</name>
            <synopsis>100 Mb full-duplex rate support.</synopsis>
            <typeRef>boolean</typeRef>
          </component>
          <component componentID="5">
            <name>1GB_HD</name>
            <synopsis>1 Gb half-duplex rate support.</synopsis>
            <typeRef>boolean</typeRef>
          </component>
          <component componentID="6">
            <name>1GB_FD</name>
            <synopsis>1 Gb full-duplex rate support.</synopsis>
            <typeRef>boolean</typeRef>
          </component>
          <component componentID="7">
            <name>10GB_FD</name>
            <synopsis>10 Gb full-duplex rate support.</synopsis>
            <typeRef>boolean</typeRef>
          </component>
          <component componentID="8">
```

```
        <name>40GB_FD</name>
        <synopsis>40 Gb full-duplex rate support.</synopsis>
        <typeRef>boolean</typeRef>
    </component>
    <component componentID="9">
        <name>100GB_FD</name>
        <synopsis>100 Gb full-duplex rate support.</synopsis>
        <typeRef>boolean</typeRef>
    </component>
    <component componentID="10">
        <name>1TB_FD</name>
        <synopsis>1 Tb full-duplex rate support.</synopsis>
        <typeRef>boolean</typeRef>
    </component>
    <component componentID="11">
        <name>Other</name>
        <synopsis>Other rate, not listed.</synopsis>
        <typeRef>boolean</typeRef>
    </component>
</struct>
</component>
<component componentID="2">
    <name>MediumConnected</name>
    <synopsis>Medium Connected to the port</synopsis>
    <struct>
        <component componentID="1">
            <name>Copper</name>
            <synopsis>Copper Medium</synopsis>
            <typeRef>boolean</typeRef>
        </component>
        <component componentID="2">
            <name>Fiber</name>
            <synopsis>Fiber Medium</synopsis>
            <typeRef>boolean</typeRef>
        </component>
    </struct>
</component>
<component componentID="3">
    <name>Auto</name>
    <synopsis>Auto-negotiation</synopsis>
    <typeRef>boolean</typeRef>
</component>
<component componentID="4">
    <name>PauseTypes</name>
    <synopsis>Pause Types supported of the port</synopsis>
    <struct>
        <component componentID="1">
            <name>Pause</name>
```

```
        <synopsis>Pause</synopsis>
        <typeRef>boolean</typeRef>
    </component>
    <component componentID="2">
        <name>AsymmetricPause</name>
        <synopsis>Asymmetric pause</synopsis>
        <typeRef>boolean</typeRef>
    </component>
</struct>
</component>
</struct>
</dataTypeDef>
<dataTypeDef>
    <name>PortCounterType</name>
    <synopsis>Counter per port</synopsis>
    <struct>
        <component componentID="1">
            <name>ReceivedPackets</name>
            <synopsis>Packets Received</synopsis>
            <typeRef>uint64</typeRef>
        </component>
        <component componentID="2">
            <name>TransmittedPackets</name>
            <synopsis>Packets Transmitted</synopsis>
            <typeRef>uint64</typeRef>
        </component>
        <component componentID="3">
            <name>ReceivedBytes</name>
            <synopsis>Bytes Received</synopsis>
            <typeRef>uint64</typeRef>
        </component>
        <component componentID="4">
            <name>TransmittedBytes</name>
            <synopsis>Bytes Transmitted</synopsis>
            <typeRef>uint64</typeRef>
        </component>
        <component componentID="5">
            <name>ReceivedDrops</name>
            <synopsis>Drops Received</synopsis>
            <typeRef>uint64</typeRef>
        </component>
        <component componentID="6">
            <name>TransmitDrops</name>
            <synopsis>Transmit Drops</synopsis>
            <typeRef>uint64</typeRef>
        </component>
        <component componentID="7">
            <name>RecieveErrors</name>
```

```

        <synopsis>Errors in reception</synopsis>
        <typeRef>uint64</typeRef>
    </component>
    <component componentID="8">
        <name>TransmitErrors</name>
        <synopsis>Errors in transmission</synopsis>
        <typeRef>uint64</typeRef>
    </component>
    <component componentID="9">
        <name>ReceivedFrameAlignmentErrors</name>
        <synopsis>Frame Alignment Errors received</synopsis>
        <typeRef>uint64</typeRef>
    </component>
    <component componentID="10">
        <name>ReceiveOverrunErrors</name>
        <synopsis>Received Overrun Errors</synopsis>
        <typeRef>uint64</typeRef>
    </component>
    <component componentID="11">
        <name>ReceivedCRCErrors</name>
        <synopsis>CRC Errors in received packets</synopsis>
        <typeRef>uint64</typeRef>
    </component>
    <component componentID="12">
        <name>Collisions</name>
        <synopsis>Collisions</synopsis>
        <typeRef>uint64</typeRef>
    </component>
</struct>
</dataTypeDef>
<!-- Data Type definitions for Queues -->
<dataTypeDef>
    <name>QueuePropertyType</name>
    <synopsis>Property type for a queue</synopsis>
    <atomic>
        <baseType>uint16</baseType>
        <specialValues>
            <specialValue value="0">
                <name>None</name>
                <synopsis>No property defined</synopsis>
            </specialValue>
            <specialValue value="1">
                <name>MinimumRate</name>
                <synopsis>Minimum datarate guaranteed</synopsis>
            </specialValue>
        </specialValues>
    </atomic>
</dataTypeDef>

```

```
<dataTypeDef>
  <name>QueueArrayPropertiesType</name>
  <synopsis>Type Definition for property</synopsis>
  <struct>
    <component componentID="1">
      <name>Property</name>
      <synopsis>One of Queue Priority Types</synopsis>
      <typeRef>QueuePropertyType</typeRef>
    </component>
    <component componentID="2">
      <name>Length</name>
      <synopsis>Length of property</synopsis>
      <typeRef>uint32</typeRef>
    </component>
  </struct>
</dataTypeDef>
<dataTypeDef>
  <name>QueueCounterType</name>
  <synopsis>Counters per queue</synopsis>
  <struct>
    <component componentID="1">
      <name>TransmitPackets</name>
      <synopsis>Packets Transmitted</synopsis>
      <typeRef>uint64</typeRef>
    </component>
    <component componentID="2">
      <name>TransmitBytes</name>
      <synopsis>Bytes Transmitted</synopsis>
      <typeRef>uint64</typeRef>
    </component>
    <component componentID="3">
      <name>TransimtOverrunErrors</name>
      <synopsis>Overrun Errors</synopsis>
      <typeRef>uint64</typeRef>
    </component>
  </struct>
</dataTypeDef>
</dataTypeDefs>
<metadataDefs>
  <metadataDef>
    <name>IngressPort</name>
    <synopsis>The Ingress port the packet has arrived from</synopsis>
    <metadataID>1024</metadataID>
    <typeRef>uint32</typeRef>
  </metadataDef>
  <metadataDef>
    <name>InPhyPort</name>
    <synopsis>The Port Index of the Physical interface the frame
```



```
    entered the switch</synopsis>
    <metadataID>1025</metadataID>
    <typeRef>uint32</typeRef>
</metadataDef>
<metadataDef>
    <name>PacketID</name>
    <synopsis>The PacketID metadata is used to uniquely identify a
    packet within the Flow Table or the Group Table to continue
    processing it after it has been returned from an OFActionLFB.
    </synopsis>
    <metadataID>1026</metadataID>
    <typeRef>uint32</typeRef>
</metadataDef>
<metadataDef>
    <name>ActionIndex</name>
    <synopsis>The Action Index metadata is used to point the row in
    the array in an Action LFB </synopsis>
    <metadataID>1027</metadataID>
    <typeRef>uint32</typeRef>
</metadataDef>
<metadataDef>
    <name>GroupIndex</name>
    <synopsis>The Group index metadata is used to point to the row of
    the array in an Group LFB</synopsis>
    <metadataID>1028</metadataID>
    <typeRef>uint32</typeRef>
</metadataDef>
<metadataDef>
    <name>LFBClassIDMetadata</name>
    <synopsis>The LFBClassID</synopsis>
    <metadataID>1029</metadataID>
    <typeRef>uint32</typeRef>
</metadataDef>
<metadataDef>
    <name>QueueIDMetadata</name>
    <synopsis>The Queue ID</synopsis>
    <metadataID>1030</metadataID>
    <typeRef>uint32</typeRef>
</metadataDef>
<metadataDef>
    <name>BufferID</name>
    <synopsis>The Buffer ID of a stored packet in the switch required
    for the PacketOut message</synopsis>
    <metadataID>1031</metadataID>
    <typeRef>uint32</typeRef>
</metadataDef>
<metadataDef>
    <name>RedirectReason</name>
```

```

    <synopsis>The reason the packet was redirected to the controller
  </synopsis>
  <metadataID>1032</metadataID>
  <atomic>
    <baseType>uchar</baseType>
    <specialValues>
      <specialValue value="0">
        <name>NoMatch</name>
        <synopsis>No match on the Flow Table (table miss)
      </synopsis>
      </specialValue>
      <specialValue value="1">
        <name>Action</name>
        <synopsis>Specific Output to controller action.</synopsis>
      </specialValue>
    </specialValues>
  </atomic>
</metadataDef>
<metadataDef>
  <name>FlowTableID</name>
  <synopsis>The FlowTable ID the packet was sent to the controller
  from</synopsis>
  <metadataID>1033</metadataID>
  <typeRef>uchar</typeRef>
</metadataDef>
<metadataDef>
  <name>ActionListMetadata</name>
  <synopsis>The Action List that may come along with the packet in
  a PacketOut message</synopsis>
  <metadataID>1034</metadataID>
  <typeRef>octetstring</typeRef>
</metadataDef>
</metadataDefs>
<LFBClassDefs>
  <!-- OpenFlow Switch LFB -->
  <LFBClassDef LFBClassID="1024">
    <name>OFSwitch</name>
    <synopsis>Similar to the FEProtocol and FEObject LFB, the
    OpenFlowSwitch LFB contains information required for the OpenFlow
    protocol.</synopsis>
    <version>1.1</version>
    <components>
      <component componentID="1" access="read-only">
        <name>DatapathID</name>
        <synopsis>Datapath unique ID. The lower 48-bits are for a MAC
        address, while the upper 16-bits are implementer-defined.
      </synopsis>
      <typeRef>uint64</typeRef>
    </components>
  </LFBClassDef>
</LFBClassDefs>

```

```
</component>
<component componentID="4" access="read-write">
  <name>MissSendLen</name>
  <synopsis>Max bytes of new flow that datapath should send to
the controller.</synopsis>
  <typeRef>uint16</typeRef>
</component>
<component componentID="5" access="read-write">
  <name>HandleFragments</name>
  <synopsis>if true drop fragments. If false no special
handling.</synopsis>
  <typeRef>boolean</typeRef>
</component>
<component componentID="6" access="read-write">
  <name>ReassembleFragments</name>
  <synopsis>If true, reassemble fragments</synopsis>
  <typeRef>boolean</typeRef>
</component>
<component componentID="7" access="read-write">
  <name>InvalidTTLtoController</name>
  <synopsis>Send packets with invalid TTL ie. 0 or 1 to
controller</synopsis>
  <typeRef>boolean</typeRef>
</component>
<component componentID="8" access="read-only">
  <name>SwitchDescription</name>
  <synopsis>Information about the switch</synopsis>
  <typeRef>SwitchDescriptionType</typeRef>
</component>
<component componentID="9" access="read-write">
  <name>Ports</name>
  <synopsis>The Ports that this switch has. It is an array of
the Port Numbers</synopsis>
  <array>
    <typeRef>uint32</typeRef>
  </array>
</component>
</components>
<capabilities>
  <capability componentID="31">
    <name>FlowStatistics</name>
    <synopsis>Whether the switch keep flow statistics</synopsis>
    <typeRef>boolean</typeRef>
  </capability>
  <capability componentID="32">
    <name>TableStatistics</name>
    <synopsis>Whether the switch keep table statistics</synopsis>
    <typeRef>boolean</typeRef>
  </capability>
</capabilities>
```

```
</capability>
<capability componentID="33">
  <name>PortStatistics</name>
  <synopsis>Whether the switch keep port statistics</synopsis>
  <typeRef>boolean</typeRef>
</capability>
<capability componentID="34">
  <name>GroupStatistics</name>
  <synopsis>Whether the switch keep group statistics</synopsis>
  <typeRef>boolean</typeRef>
</capability>
<capability componentID="35">
  <name>IPReassembly</name>
  <synopsis>Whether the switch can reassemble IP fragments
  </synopsis>
  <typeRef>boolean</typeRef>
</capability>
<capability componentID="36">
  <name>QueueStats</name>
  <synopsis>Whether the switch keeps queue statistics
  </synopsis>
  <typeRef>boolean</typeRef>
</capability>
<capability componentID="37">
  <name>ARPMatchIP</name>
  <synopsis>Whether the switch matches IP addresses in APR
  packets</synopsis>
  <typeRef>boolean</typeRef>
</capability>
<capability componentID="38">
  <name>ActionsSupported</name>
  <synopsis>What actions are supported</synopsis>
  <array>
    <atomic>
      <baseType>ActionType</baseType>
      <rangeRestriction>
        <allowedRange max="65534" min="0"/>
      </rangeRestriction>
    </atomic>
    <contentKey contentKeyID="1">
      <contentKeyField>ActionType</contentKeyField>
    </contentKey>
  </array>
</capability>
<capability componentID="39">
  <name>MaxBufferedPackets</name>
  <synopsis>Maximum packets buffered at once.</synopsis>
  <typeRef>uint32</typeRef>
```

```
</capability>
<capability componentID="40">
  <name>TablesSupported</name>
  <synopsis>Number of tables supported by the datapath
</synopsis>
  <typeRef>uchar</typeRef>
</capability>
</capabilities>
<events baseID="61">
  <event eventID="1">
    <name>PortAdded</name>
    <synopsis>This event is sent when a port is added</synopsis>
    <eventTarget>
      <eventField>Ports</eventField>
    </eventTarget>
    <eventCreated/>
  </event>
  <event eventID="2">
    <name>PortDeleted</name>
    <synopsis>This event is sent when a port is deleted
</synopsis>
    <eventTarget>
      <eventField>Ports</eventField>
    </eventTarget>
    <eventDeleted/>
  </event>
  <event eventID="3">
    <name>PortModified</name>
    <synopsis>This event is sent when a port is modified
</synopsis>
    <eventTarget>
      <eventField>Ports</eventField>
    </eventTarget>
    <eventChanged/>
  </event>
</events>
</LFBClassDef>
<!--FlowTable LFB -->
<LFBClassDef LFBClassID="1025">
  <name>OFFlowTables</name>
  <synopsis>An LFB that houses all OpenFlow Flow Tables residing in
the switch</synopsis>
  <version>1.1</version>
  <inputPorts>
    <inputPort group="true">
      <name>InputPort</name>
      <synopsis>An Input port that expects packets from an OFPort
LFB</synopsis>
```

```
<expectation>
  <frameExpected>
    <ref>Arbitrary</ref>
  </frameExpected>
  <metadataExpected>
    <ref>IngressPort</ref>
    <ref>InPhyPort</ref>
  </metadataExpected>
</expectation>
</inputPort>
<inputPort group="true">
  <name>PacketReturn</name>
  <synopsis>A port that expects the packet to be returned from
an OFActionLFB. If the OFActionLFB is the OFQueueLFB then the
QueueID metadata is expected as well.</synopsis>
  <expectation>
    <frameExpected>
      <ref>Arbitrary</ref>
    </frameExpected>
    <metadataExpected>
      <ref>PacketID</ref>
      <ref dependency="optional">QueueID</ref>
    </metadataExpected>
  </expectation>
</inputPort>
<inputPort group="false">
  <name>RedirectPacketIn</name>
  <synopsis>A port that expects a packet from the controller
</synopsis>
  <expectation>
    <frameExpected>
      <ref>Arbitrary</ref>
    </frameExpected>
    <metadataExpected>
      <ref>ActionList</ref>
      <ref>IngressPort</ref>
    </metadataExpected>
  </expectation>
</inputPort>
<inputPort group="false">
  <name>RedirectBufferIn</name>
  <synopsis>A port that expects a Buffer ID index from the
controller</synopsis>
  <expectation>
    <metadataExpected>
      <ref>BufferID</ref>
      <ref>ActionList</ref>
      <ref>IngressPort</ref>
    </metadataExpected>
  </expectation>
</inputPort>
```

```

        </metadataExpected>
    </expectation>
</inputPort>
</inputPorts>
<outputPorts>
  <outputPort group="true">
    <name>OutputPort</name>
    <synopsis>A port that produces packets leaving the flow table
    to go to the OFOutputAction (to be sent to an output port)
    </synopsis>
    <product>
      <frameProduced>
        <ref>Arbitrary</ref>
      </frameProduced>
      <metadataProduced>
        <one-of>
          <metadataSet>
            <ref>IngressPort</ref>
            <ref>InPhyPort</ref>
          </metadataSet>
          <metadataSet>
            <ref>IngressPort</ref>
            <ref>InPhyPort</ref>
            <ref>QueueID</ref>
          </metadataSet>
        </one-of>
      </metadataProduced>
    </product>
  </outputPort>
  <outputPort group="true">
    <name>GroupPort</name>
    <synopsis>A port that produces packets leaving the flow table
    to go to the OFGroupTable.</synopsis>
    <product>
      <frameProduced>
        <ref>Arbitrary</ref>
      </frameProduced>
      <metadataProduced>
        <one-of>
          <metadataSet>
            <ref>IngressPort</ref>
            <ref>InPhyPort</ref>
            <ref>GroupIndex</ref>
          </metadataSet>
          <metadataSet>
            <ref>IngressPort</ref>
            <ref>InPhyPort</ref>
            <ref>QueueID</ref>
          </metadataSet>
        </one-of>
      </metadataProduced>
    </product>
  </outputPort>
</outputPorts>

```

```

        <ref>GroupIndex</ref>
      </metadataSet>
    </one-of>
  </metadataProduced>
</product>
</outputPort>
<outputPort group="true">
  <name>ActionPort</name>
  <synopsis>A port that sends the packet to an OFActionLFB
</synopsis>
  <product>
    <frameProduced>
      <ref>Arbitrary</ref>
    </frameProduced>
    <metadataProduced>
      <one-of>
        <metadataSet>
          <ref>LFBClassIDMetadata</ref>
          <ref>PacketID</ref>
        </metadataSet>
        <metadataSet>
          <ref>LFBClassIDMetadata</ref>
          <ref>PacketID</ref>
          <ref>ActionIndex</ref>
        </metadataSet>
      </one-of>
    </metadataProduced>
  </product>
</outputPort>
<outputPort group="false">
  <name>RedirectPacketOut</name>
  <synopsis>A port that sends a packet or part of it to the
controller</synopsis>
  <product>
    <frameProduced>
      <ref>Arbitrary</ref>
    </frameProduced>
    <metadataProduced>
      <one-of>
        <metadataSet>
          <ref>IngressPort</ref>
          <ref>InPhyPort</ref>
          <ref>RedirectReason</ref>
          <ref>FlowTableID</ref>
        </metadataSet>
        <metadataSet>
          <ref>IngressPort</ref>
          <ref>InPhyPort</ref>

```



```

        <ref>RedirectReason</ref>
        <ref>FlowTableID</ref>
        <ref>BufferID</ref>
    </metadataSet>
</one-of>
</metadataProduced>
</product>
</outputPort>
</outputPorts>
<components>
  <component componentID="1" access="read-write">
    <name>FlowTables</name>
    <synopsis>Flow entries inside the FlowTable LFB</synopsis>
    <array>
      <struct>
        <component componentID="1">
          <name>FlowEntries</name>
          <synopsis>An array of Flow Entries</synopsis>
          <array>
            <typeRef>FlowEntry</typeRef>
          </array>
        </component>
        <component componentID="2">
          <name>FlowTableCounter</name>
          <synopsis>A counter for each Flow Table</synopsis>
          <typeRef>TableCounterType</typeRef>
        </component>
        <component componentID="3">
          <name>MissBehaviour</name>
          <synopsis>What should the FlowTable do if a miss occurs</synopsis>
          <typeRef>FlowTableMissConfigType</typeRef>
        </component>
      </struct>
    </array>
  </component>
  <component componentID="2" access="read-write">
    <name>ApplyActionList</name>
    <synopsis>Table of actions for each flow</synopsis>
    <array>
      <typeRef>Actions</typeRef>
    </array>
  </component>
  <component componentID="3" access="read-write">
    <name>WriteActions</name>
    <synopsis>Table of Actions to write to the ActionSet</synopsis>
    <array>

```

```

        <typeRef>Actions</typeRef>
    </array>
</component>
<component componentID="4" access="read-write">
    <name>WriteMetadataTable</name>
    <synopsis>The write MetaDataTable</synopsis>
    <array>
        <typeRef>WriteMetadataTableType</typeRef>
    </array>
</component>
<component componentID="5" access="read-write">
    <name>GotoFlowTable</name>
    <synopsis>Containing the FlowTable IDs this flow should go
to.</synopsis>
    <array>
        <typeRef>uint32</typeRef>
    </array>
</component>
<component componentID="6" access="read-write">
    <name>GroupTable</name>
    <synopsis>Table of group indeces to point a packet to
</synopsis>
    <array>
        <typeRef>uint32</typeRef>
    </array>
</component>
</components>
<events baseID="61">
    <event eventID="1">
        <name>FlowRemoved</name>
        <synopsis>If a CE subscribes to this event, it will send an
event when a flow is removed.</synopsis>
        <eventTarget>
            <eventField>FlowEntries</eventField>
            <eventSubscript>FlowEntry</eventSubscript>
        </eventTarget>
        <eventDeleted/>
        <eventReports>
            <eventReport>
                <eventField>FlowTableID</eventField>
            </eventReport>
            <eventReport>
                <eventField>FlowEntries</eventField>
                <eventSubscript>FlowEntry</eventSubscript>
                <eventField>Cookie</eventField>
            </eventReport>
            <eventReport>
                <eventField>FlowEntries</eventField>

```

```

        <eventSubscript>FlowEntry</eventSubscript>
        <eventField>MatchFields</eventField>
    </eventReport>
    <eventReport>
        <eventField>FlowEntries</eventField>
        <eventSubscript>FlowEntry</eventSubscript>
        <eventField>Timeouts</eventField>
        <eventSubscript>IdleTimeout</eventSubscript>
    </eventReport>
    <eventReport>
        <eventField>FlowEntries</eventField>
        <eventSubscript>FlowEntry</eventSubscript>
        <eventField>Priority</eventField>
    </eventReport>
</eventReports>
</event>
</events>
</LFBClassDef>
<!-- GroupTable LFB -->
<LFBClassDef LFBClassID="1026">
    <name>OFGroupTable</name>
    <synopsis>The OpenFlow Group Tables</synopsis>
    <version>1.1</version>
    <inputPorts>
        <inputPort group="true">
            <name>PacketIn</name>
            <synopsis>A port to expect packets, the GroupIndex metadata,
            the IngressPort and InPhyPort and optionally the QueueID if
            the packet has already been through the OFActionSetQueue LFB.
            </synopsis>
            <expectation>
                <frameExpected>
                    <ref>Arbitrary</ref>
                </frameExpected>
                <metadataExpected>
                    <ref>GroupIndex</ref>
                    <ref>IngressPort</ref>
                    <ref>InPhyPort</ref>
                    <ref dependency="optional">QueueID</ref>
                </metadataExpected>
            </expectation>
        </inputPort>
        <inputPort group="true">
            <name>PacketReturn</name>
            <synopsis>A port that expects the packet to be returned from
            an OFActionLFB. If the OFActionLFB is the OFQueueLFB then the
            QueueID metadata is expected as well.</synopsis>
            <expectation>

```

```

        <frameExpected>
          <ref>Arbitrary</ref>
        </frameExpected>
        <metadataExpected>
          <ref>PacketID</ref>
          <ref dependency="optional">QueueID</ref>
        </metadataExpected>
      </expectation>
    </inputPort>
  </inputPorts>
  <outputPorts>
    <outputPort group="true">
      <name>PacketOut</name>
      <synopsis>The port to return the packet to caller</synopsis>
      <product>
        <frameProduced>
          <ref>Arbitrary</ref>
        </frameProduced>
      </product>
    </outputPort>
    <outputPort group="true">
      <name>ActionPort</name>
      <synopsis>A port that sends the packet to an OFActionLFB
      </synopsis>
      <product>
        <frameProduced>
          <ref>Arbitrary</ref>
        </frameProduced>
        <metadataProduced>
          <one-of>
            <metadataSet>
              <ref>LFBClassIDMetadata</ref>
              <ref>LFBInstanceIDMetadata</ref>
            </metadataSet>
            <metadataSet>
              <ref>LFBClassIDMetadata</ref>
              <ref>LFBInstanceIDMetadata</ref>
              <ref>ActionIndex</ref>
            </metadataSet>
          </one-of>
        </metadataProduced>
      </product>
    </outputPort>
  </outputPorts>
  <components>
    <component componentID="1">
      <name>GroupTable</name>
      <synopsis>The group table</synopsis>
    </component>
  </components>

```

```

        <array>
          <typeRef>GroupTableEntry</typeRef>
        </array>
      </component>
    </components>
  </LFBClassDef>
  <!-- Port LFB -->
  <LFBClassDef LFBClassID="1027">
    <name>OFPort</name>
    <synopsis>Input or Output port of an OpenFlow switch</synopsis>
    <version>1.1</version>
    <inputPorts>
      <inputPort group="false">
        <name>PacketOut</name>
        <synopsis>The input port of the Port LFB from the
          OFActionOutput LFB to output packets.</synopsis>
        <expectation>
          <frameExpected>
            <ref>Arbitrary</ref>
          </frameExpected>
          <metadataExpected>
            <ref dependency="optional">QueueID</ref>
          </metadataExpected>
        </expectation>
      </inputPort>
    </inputPorts>
    <outputPorts>
      <outputPort group="false">
        <name>PacketIn</name>
        <synopsis>Sends a packet to the OFFlowTables that is received
          by the OFPort LFB.</synopsis>
        <product>
          <frameProduced>
            <ref>Arbitrary</ref>
          </frameProduced>
          <metadataProduced>
            <ref>IngressPort</ref>
            <ref>InPhyPort</ref>
          </metadataProduced>
        </product>
      </outputPort>
      <outputPort group="true">
        <name>QueueOut</name>
        <synopsis>Sends a packet to the OFQueue LFB to be processed
          and output from the switch</synopsis>
        <product>
          <frameProduced>
            <ref>Arbitrary</ref>

```

```
        </frameProduced>
    </product>
</outputPort>
</outputPorts>
<components>
  <component componentID="1" access="read-only">
    <name>PortNumber</name>
    <synopsis>The port number uniquely identifies a port within a
      switch.</synopsis>
    <typeRef>PortNumberType</typeRef>
  </component>
  <component componentID="2" access="read-only">
    <name>IEEEMAC</name>
    <synopsis>MAC Address of the port</synopsis>
    <typeRef>IEEEMAC</typeRef>
  </component>
  <component componentID="3" access="read-only">
    <name>Name</name>
    <synopsis>Human readable name of the port</synopsis>
    <typeRef>string[16]</typeRef>
  </component>
  <component componentID="4" access="read-write">
    <name>Configuration</name>
    <synopsis>Configuration of the port</synopsis>
    <typeRef>PortConfigurationType</typeRef>
  </component>
  <component componentID="5" access="read-only">
    <name>State</name>
    <synopsis>State of the OpenFlow Switch</synopsis>
    <typeRef>PortState</typeRef>
  </component>
  <component componentID="6" access="read-only">
    <name>CurrentFeatures</name>
    <synopsis>Current features of the port</synopsis>
    <typeRef>PortFeaturesType</typeRef>
  </component>
  <component componentID="7" access="read-write">
    <name>Advertised</name>
    <synopsis>Features advertised by the port</synopsis>
    <typeRef>PortFeaturesType</typeRef>
  </component>
  <component componentID="8" access="read-only">
    <name>CurrentSpeed</name>
    <synopsis>Current port bitrate in kbps</synopsis>
    <typeRef>uint32</typeRef>
  </component>
  <component componentID="9" access="read-only">
    <name>MaximumSpeed</name>
```

```

        <synopsis>Maximum port bitrate in kbps</synopsis>
        <typeRef>uint32</typeRef>
    </component>
    <component componentID="10" access="read-only">
        <name>PortCounter</name>
        <synopsis>Counters for the port</synopsis>
        <typeRef>PortCounterType</typeRef>
    </component>
</components>
<capabilities>
    <capability componentID="31">
        <name>Supported</name>
        <synopsis>Features Supported by the port</synopsis>
        <typeRef>PortFeaturesType</typeRef>
    </capability>
    <capability componentID="32">
        <name>Peer</name>
        <synopsis>Features advertised by the peer</synopsis>
        <typeRef>PortFeaturesType</typeRef>
    </capability>
</capabilities>
</LFBClassDef>
<!-- Queue LFB -->
<LFBClassDef LFBClassID="1028">
    <name>OFQueue</name>
    <synopsis>A queue LFB that can be attached to a port and be used
    to map flows on it. Flows mapped to a queue will be treated
    according to that queue's configuration</synopsis>
    <version>1.1</version>
    <inputPorts>
        <inputPort group="true">
            <name>PacketIn</name>
            <synopsis>An input port that expects any kind of frame.
            </synopsis>
            <expectation>
                <frameExpected>
                    <ref>Arbitrary</ref>
                </frameExpected>
            </expectation>
        </inputPort>
    </inputPorts>
    <components>
        <component componentID="1" access="read-only">
            <name>QueueID</name>
            <synopsis>ID for the specific queue</synopsis>
            <typeRef>uint32</typeRef>
        </component>
        <component componentID="2" access="read-write">

```

```
<name>Properties</name>
<synopsis>List of queue properties</synopsis>
<array>
  <typeRef>QueueArrayPropertiesType</typeRef>
</array>
</component>
<component componentID="3" access="read-only">
  <name>QueueCounter</name>
  <synopsis>Counters for the queue</synopsis>
  <typeRef>QueueCounterType</typeRef>
</component>
</components>
</LFBClassDef>
<!-- OFRedirectIn LFB -->
<LFBClassDef LFBClassID="1029">
  <name>OFRedirectIn</name>
  <synopsis>The OFRedirectIn LFB abstracts the process for the
controller to inject data packets into the switch to input data
packets into the data path.</synopsis>
<version>1.1</version>
<derivedFrom>RedirectIn</derivedFrom>
<outputPorts>
  <outputPort group="false">
    <name>PacketIn</name>
    <synopsis>An output port that sends a packet in the data
path</synopsis>
    <product>
      <frameProduced>
        <ref>Arbitrary</ref>
      </frameProduced>
      <metadataProduced>
        <ref>ActionList</ref>
        <ref>IngressPort</ref>
      </metadataProduced>
    </product>
  </outputPort>
  <outputPort group="false">
    <name>BufferIn</name>
    <synopsis>An output port that sends only the buffer id to
locate a buffered packet</synopsis>
    <product>
      <frameProduced>
        <ref>Null</ref>
      </frameProduced>
      <metadataProduced>
        <ref>BufferID</ref>
        <ref>ActionList</ref>
        <ref>IngressPort</ref>
      </metadataProduced>
    </product>
  </outputPort>
</outputPorts>
</LFBClassDef>
```



```

        </metadataProduced>
    </product>
</outputPort>
</outputPorts>
</LFBClassDef>
<!-- OFRedirectOut LFB -->
<LFBClassDef LFBClassID="1030">
    <name>OFRedirectOut</name>
    <synopsis>The OFRedirectOut LFB abstracts the process for the
switch to deliver data packets to the controller</synopsis>
    <version>1.1</version>
    <inputPorts>
        <inputPort group="true">
            <name>Outgoing</name>
            <synopsis>The input port expects either the whole packet to
be sent to the controller or part of it along with the buffer
ID</synopsis>
            <expectation>
                <frameExpected>
                    <ref>Arbitrary</ref>
                </frameExpected>
                <metadataExpected>
                    <ref>IngressPort</ref>
                    <ref>InPhyPort</ref>
                    <ref>RedirectReason</ref>
                    <ref dependency="optional">FlowTableID</ref>
                    <ref dependency="optional">BufferID</ref>
                </metadataExpected>
            </expectation>
        </inputPort>
    </inputPorts>
</LFBClassDef>
<!-- Action LFBs -->
<LFBClassDef LFBClassID="1031">
    <name>OFAction</name>
    <synopsis>An LFB that performs one specific action on a packet in
the OpenFlow switch. The OFActionLFB expects any kind of packet
and as metadata the FlowTableInstanceID to know from which Flow
Table the packet has arrived from and the Action Index to specify
the row in the Action Table, if there is an Action table.
</synopsis>
    <version>1.1</version>
    <inputPorts>
        <inputPort group="true">
            <name>PacketIn</name>
            <synopsis>An input port that gets the packet to perform the
action on. Expects the ClassID of the LFB that calls it to
know to which LFB to return it to. Can accept calls from the

```

```

OFFlowTables or the OFGroupLFB.</synopsis>
<expectation>
  <frameExpected>
    <ref>Arbitrary</ref>
  </frameExpected>
  <metadataExpected>
    <ref>PacketID</ref>
    <ref>LFBClassIDMetadata</ref>
    <ref dependency="optional">ActionIndex</ref>
    <ref dependency="optional">IngressPort</ref>
    <ref dependency="optional">InPhyPort</ref>
    <ref dependency="optional">QueueID</ref>
  </metadataExpected>
</expectation>
</inputPort>
</inputPorts>
<outputPorts>
  <outputPort group="true">
    <name>PacketOut</name>
    <synopsis>The output port from which the packet will be send
back to the LFB (OFFlowTables or OFGroupTable) from which it
came from.</synopsis>
    <product>
      <frameProduced>
        <ref>Arbitrary</ref>
      </frameProduced>
      <metadataProduced>
        <ref>PacketID</ref>
      </metadataProduced>
    </product>
  </outputPort>
</outputPorts>
</LFBClassDef>
<LFBClassDef LFBClassID="1032">
  <name>OFActionOutput</name>
  <synopsis>An LFB that performs the Output Action</synopsis>
  <version>1.1</version>
  <derivedFrom>OFAction</derivedFrom>
  <outputPorts>
    <outputPort group="true">
      <name>PortOutput</name>
      <synopsis>Send a copy of the packet to the specified port
</synopsis>
      <product>
        <frameProduced>
          <ref>Arbitrary</ref>
        </frameProduced>
      </product>
    </outputPort>
  </outputPorts>
</LFBClassDef>

```

```
</outputPort>
<outputPort group="true">
  <name>All</name>
  <synopsis>Send the packet out all standard ports, but not to
the ingress port or ports configured not to forward
</synopsis>
  <product>
    <frameProduced>
      <ref>Arbitrary</ref>
    </frameProduced>
  </product>
</outputPort>
<outputPort group="false">
  <name>Controller</name>
  <synopsis>Send the packet to the controller</synopsis>
  <product>
    <frameProduced>
      <ref>Arbitrary</ref>
    </frameProduced>
  </product>
</outputPort>
<outputPort group="false">
  <name>Table</name>
  <synopsis>Submit the packet to the first flow table so that
the packet can be processed through the regular OpenFlow
pipeline. Only valid in the action set of a packet-out
message</synopsis>
  <product>
    <frameProduced>
      <ref>Arbitrary</ref>
    </frameProduced>
  </product>
</outputPort>
<outputPort group="true">
  <name>InPort</name>
  <synopsis>Send the packet out the ingress port.</synopsis>
  <product>
    <frameProduced>
      <ref>Arbitrary</ref>
    </frameProduced>
  </product>
</outputPort>
<outputPort group="false">
  <name>Local</name>
  <synopsis>Send the packet to the switch's local networking
stack</synopsis>
  <product>
    <frameProduced>
```

```

        <ref>Arbitrary</ref>
      </frameProduced>
    </product>
  </outputPort>
  <outputPort group="false">
    <name>Normal</name>
    <synopsis>Process the packet using the traditional
non-OpenFlow pipeline of the switch.</synopsis>
    <product>
      <frameProduced>
        <ref>Arbitrary</ref>
      </frameProduced>
    </product>
  </outputPort>
  <outputPort group="true">
    <name>Flood</name>
    <synopsis>Flood the packet using the normal pipeline of the
switch.</synopsis>
    <product>
      <frameProduced>
        <ref>Arbitrary</ref>
      </frameProduced>
    </product>
  </outputPort>
</outputPorts>
<components>
  <component componentID="1" access="read-write">
    <name>OutputActionTable</name>
    <synopsis>Output to switch port</synopsis>
    <array>
      <struct>
        <component componentID="1">
          <name>Port</name>
          <synopsis>The port to send the packet out</synopsis>
          <typeRef>PortNumberType</typeRef>
        </component>
        <component componentID="2">
          <name>MaxLength</name>
          <synopsis>If the port is the controller sets the
maximum number of bytes to send.</synopsis>
          <typeRef>uint16</typeRef>
        </component>
      </struct>
    </array>
  </component>
</components>
</LFBClassDef>
<LFBClassDef LFBClassID="1033">

```

```
<name>OFActionSetVLANVID</name>
<synopsis>An LFB that performs the Set VLANID Action</synopsis>
<version>1.1</version>
<derivedFrom>OFAction</derivedFrom>
<components>
  <component componentID="1" access="read-write">
    <name>SetVLANVIDActionTable</name>
    <synopsis>Set the 802.1q VLAN ID</synopsis>
    <array>
      <typeRef>uint16</typeRef>
    </array>
  </component>
</components>
</LFBClassDef>
<LFBClassDef LFBClassID="1034">
  <name>OFActionSetVLANPriority</name>
  <synopsis>An LFB that performs the Set VLAN Priority Action
  </synopsis>
  <version>1.1</version>
  <derivedFrom>OFAction</derivedFrom>
  <components>
    <component componentID="1" access="read-write">
      <name>SetVLANPriorityActionTable</name>
      <synopsis>Set the 802.1q VLAN Priority</synopsis>
      <array>
        <typeRef>uchar</typeRef>
      </array>
    </component>
  </components>
</LFBClassDef>
<LFBClassDef LFBClassID="1035">
  <name>OFActionSetMACSource</name>
  <synopsis>An LFB that performs the Set MAC Source Action
  </synopsis>
  <version>1.1</version>
  <derivedFrom>OFAction</derivedFrom>
  <components>
    <component componentID="1" access="read-write">
      <name>SetMACSourceActionTable</name>
      <synopsis>Set MAC source address</synopsis>
      <array>
        <typeRef>IEEEMAC</typeRef>
      </array>
    </component>
  </components>
</LFBClassDef>
<LFBClassDef LFBClassID="1036">
  <name>OFActionSetMACDestination</name>
```

```
<synopsis>An LFB that performs the Set MAC Destination Action
</synopsis>
<version>1.1</version>
<derivedFrom>OFAction</derivedFrom>
<components>
  <component componentID="1" access="read-write">
    <name>SetMACDestinationActionTable</name>
    <synopsis>Set MAC destination address</synopsis>
    <array>
      <typeRef>IEEEMAC</typeRef>
    </array>
  </component>
</components>
</LFBClassDef>
<LFBClassDef LFBClassID="1037">
  <name>OFActionSetIPSource</name>
  <synopsis>An LFB that performs the Set IP Source Action
  </synopsis>
  <version>1.1</version>
  <derivedFrom>OFAction</derivedFrom>
  <components>
    <component componentID="1" access="read-write">
      <name>SetIPSourceActionTable</name>
      <synopsis>Set the IP source address</synopsis>
      <array>
        <typeRef>IPv4Addr</typeRef>
      </array>
    </component>
  </components>
</LFBClassDef>
<LFBClassDef LFBClassID="1038">
  <name>OFActionSetIPDestination</name>
  <synopsis>An LFB that performs the Set IP Destination Action
  </synopsis>
  <version>1.1</version>
  <derivedFrom>OFAction</derivedFrom>
  <components>
    <component componentID="1" access="read-write">
      <name>SetIPDestinationActionTable</name>
      <synopsis>Set the IP destination address</synopsis>
      <array>
        <typeRef>IPv4Addr</typeRef>
      </array>
    </component>
  </components>
</LFBClassDef>
<LFBClassDef LFBClassID="1039">
  <name>OFActionSetIPTOS</name>
```

```
<synopsis>An LFB that performs the Set VLANID Action</synopsis>
<version>1.1</version>
<derivedFrom>OFAction</derivedFrom>
<components>
  <component componentID="1" access="read-write">
    <name>SetIPTOSActionTable</name>
    <synopsis>Set IP ToS field</synopsis>
    <array>
      <typeRef>uchar</typeRef>
    </array>
  </component>
</components>
</LFBClassDef>
<LFBClassDef LFBClassID="1040">
  <name>OFActionSetIPECN</name>
  <synopsis>An LFB that performs the Set IP ECN Action</synopsis>
  <version>1.1</version>
  <derivedFrom>OFAction</derivedFrom>
  <components>
    <component componentID="1" access="read-write">
      <name>SetIPECNActionTable</name>
      <synopsis>Set IP ECN field</synopsis>
      <array>
        <typeRef>uchar</typeRef>
      </array>
    </component>
  </components>
</LFBClassDef>
<LFBClassDef LFBClassID="1041">
  <name>OFActionSetTCPSource</name>
  <synopsis>An LFB that performs the Set TCP/UDP/SCTP Source port
  Action</synopsis>
  <version>1.1</version>
  <derivedFrom>OFAction</derivedFrom>
  <components>
    <component componentID="1" access="read-write">
      <name>SetTCPSourceActionTable</name>
      <synopsis>Sets TCP/UDP/SCTP source port</synopsis>
      <array>
        <typeRef>uint16</typeRef>
      </array>
    </component>
  </components>
</LFBClassDef>
<LFBClassDef LFBClassID="1042">
  <name>OFActionSetTCPDestination</name>
  <synopsis>An LFB that performs the Set TCP/UDP/SCTP Destination
  port Action</synopsis>
```

```
<version>1.1</version>
<derivedFrom>OFAction</derivedFrom>
<components>
  <component componentID="1" access="read-write">
    <name>SetTCPDestinationActionTable</name>
    <synopsis>Sets TCP/UDP/SCTP destination port</synopsis>
    <array>
      <typeRef>uint16</typeRef>
    </array>
  </component>
</components>
</LFBClassDef>
<LFBClassDef LFBClassID="1043">
  <name>OFActionCopyTTLOut</name>
  <synopsis>An LFB that performs the copy TTL outwards Action
</synopsis>
  <version>1.1</version>
  <derivedFrom>OFAction</derivedFrom>
</LFBClassDef>
<LFBClassDef LFBClassID="1044">
  <name>OFActionCopyTTLIn</name>
  <synopsis>An LFB that performs the copy TTL inwards Action
</synopsis>
  <version>1.1</version>
  <derivedFrom>OFAction</derivedFrom>
</LFBClassDef>
<LFBClassDef LFBClassID="1045">
  <name>OFActionSetMPLSLabel</name>
  <synopsis>An LFB that performs the Set MPLS Label Action
</synopsis>
  <version>1.1</version>
  <derivedFrom>OFAction</derivedFrom>
  <components>
    <component componentID="1" access="read-write">
      <name>SetMPLSLabelActionTable</name>
      <synopsis>Sets MPLS Label Table</synopsis>
      <array>
        <typeRef>uint32</typeRef>
      </array>
    </component>
  </components>
</LFBClassDef>
<LFBClassDef LFBClassID="1046">
  <name>OFActionSetMPLSTC</name>
  <synopsis>An LFB that performs the Set MPLS Traffic Class Action
</synopsis>
  <version>1.1</version>
  <derivedFrom>OFAction</derivedFrom>
```



```
<components>
  <component componentID="1" access="read-write">
    <name>SetMPLSTCActionTable</name>
    <synopsis>Sets MPLS Traffic Class Table</synopsis>
    <array>
      <typeRef>uchar</typeRef>
    </array>
  </component>
</components>
</LFBClassDef>
<LFBClassDef LFBClassID="1047">
  <name>OFActionSetMPLSTTL</name>
  <synopsis>An LFB that performs the Set MPLS TTL Action</synopsis>
  <version>1.1</version>
  <derivedFrom>OFAction</derivedFrom>
  <components>
    <component componentID="1" access="read-write">
      <name>SetMPLSTTLTable</name>
      <synopsis>Sets MPLS TTL Table</synopsis>
      <array>
        <typeRef>uchar</typeRef>
      </array>
    </component>
  </components>
</LFBClassDef>
<LFBClassDef LFBClassID="1048">
  <name>OFActionDecrementMPLSTTL</name>
  <synopsis>An LFB that performs the decrementation of the MPLS TTL
  Action</synopsis>
  <version>1.1</version>
  <derivedFrom>OFAction</derivedFrom>
</LFBClassDef>
<LFBClassDef LFBClassID="1049">
  <name>OFActionPushVLAN</name>
  <synopsis>An LFB that performs the Push VLAN Action</synopsis>
  <version>1.1</version>
  <derivedFrom>OFAction</derivedFrom>
  <components>
    <component componentID="1" access="read-write">
      <name>PushVLANTable</name>
      <synopsis>Push VLAN Table</synopsis>
      <array>
        <typeRef>uint16</typeRef>
      </array>
    </component>
  </components>
</LFBClassDef>
<LFBClassDef LFBClassID="1050">
```

```

    <name>OFActionPopVLAN</name>
    <synopsis>An LFB that performs the Pop VLAN Action</synopsis>
    <version>1.1</version>
    <derivedFrom>OFAction</derivedFrom>
</LFBClassDef>
<LFBClassDef LFBClassID="1051">
    <name>OFActionPushMPLS</name>
    <synopsis>An LFB that performs the Push MPLS Action</synopsis>
    <version>1.1</version>
    <derivedFrom>OFAction</derivedFrom>
    <components>
        <component componentID="1" access="read-write">
            <name>PushMPLSTable</name>
            <synopsis>Push MPLS Table</synopsis>
            <array>
                <typeRef>uint16</typeRef>
            </array>
        </component>
    </components>
</LFBClassDef>
<LFBClassDef LFBClassID="1052">
    <name>OFActionPopMPLS</name>
    <synopsis>An LFB that performs the Pop MPLS Action</synopsis>
    <version>1.1</version>
    <derivedFrom>OFAction</derivedFrom>
    <components>
        <component componentID="1" access="read-write">
            <name>PopMPLSTable</name>
            <synopsis>Pop MPLS Table</synopsis>
            <array>
                <typeRef>uint16</typeRef>
            </array>
        </component>
    </components>
</LFBClassDef>
<LFBClassDef LFBClassID="1053">
    <name>OFActionSetQueue</name>
    <synopsis>An LFB that performs the Set Queue Action</synopsis>
    <version>1.1</version>
    <derivedFrom>OFAction</derivedFrom>
    <outputPorts>
        <outputPort group="true">
            <name>QueuePacketOut</name>
            <synopsis>The output port from which the packet will be send
            back to the Flow Table/GroupTable from which it came from.
            </synopsis>
            <product>
                <frameProduced>

```

```

        <ref>Arbitrary</ref>
      </frameProduced>
      <metadataProduced>
        <ref>PacketID</ref>
        <ref>QueueID</ref>
      </metadataProduced>
    </product>
  </outputPort>
</outputPorts>
<components>
  <component componentID="1" access="read-write">
    <name>SetQueueTable</name>
    <synopsis>Sets Queue Table</synopsis>
    <array>
      <typeRef>uint32</typeRef>
    </array>
  </component>
</components>
</LFBClassDef>
<LFBClassDef LFBClassID="1054">
  <name>OFActionSetIPTTL</name>
  <synopsis>An LFB that performs the Set IP TTL Action</synopsis>
  <version>1.1</version>
  <derivedFrom>OFAction</derivedFrom>
  <components>
    <component componentID="1" access="read-write">
      <name>SetIPTTLActionTable</name>
      <synopsis>Sets IP TTL Table</synopsis>
      <array>
        <typeRef>uchar</typeRef>
      </array>
    </component>
  </components>
</LFBClassDef>
<LFBClassDef LFBClassID="1055">
  <name>OFActionDecrementIPTTL</name>
  <synopsis>An LFB that performs the decrementation of the IP TTL
  Action</synopsis>
  <version>1.1</version>
  <derivedFrom>OFAction</derivedFrom>
</LFBClassDef>
</LFBClassDefs>
</LFBLibrary>

```

OpenFlow XML Library

## 7. Acknowledgements

The authors would like to thank Ahmad N. Quttoum, Zoltan Lajos Kis, Joel Halpern and especially Jamal Hadi Salim, for discussions which helped shape this document.

## 8. IANA Considerations

(TBD)

## 9. Security Considerations

TBD

## 10. References

### 10.1. Normative References

- [I-D.ietf-forces-lfb-lib]  
Wang, W., Haleplidis, E., Ogawa, K., Li, C., and J. Halpern, "ForCES Logical Function Block (LFB) Library", draft-ietf-forces-lfb-lib-08 (work in progress), February 2012.
- [McKeown] "McKeown, N., Anderson, T., Balakrishnan, H., et al, "OpenFlow: enabling innovation in campus networks", ACM SIGCOMM Computer Communication Review. 2008, 38(2): 69-74.", <<http://www.OpenFlow.org/documents/OpenFlow-spec-v1.1.0.pdf>>.
- [OpenFlowSpec1.1]  
<http://www.OpenFlow.org/>, "The OpenFlow 1.1 Specification.", <<http://www.OpenFlow.org/documents/OpenFlow-spec-v1.1.0.pdf>>.
- [RFC3654] Khosravi, H. and T. Anderson, "Requirements for Separation of IP Control and Forwarding", RFC 3654, November 2003.
- [RFC3746] Yang, L., Dantu, R., Anderson, T., and R. Gopal, "Forwarding and Control Element Separation (ForCES) Framework", RFC 3746, April 2004.
- [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.
- [RFC5813] Haas, R., "Forwarding and Control Element Separation (ForCES) MIB", RFC 5813, March 2010.
- [RFC6053] Haleplidis, E., Ogawa, K., Wang, W., and J. Hadi Salim, "Implementation Report for Forwarding and Control Element Separation (ForCES)", RFC 6053, November 2010.

## 10.2. Informative References

- [RFC2629] Rose, M., "Writing I-Ds and RFCs using XML", RFC 2629, June 1999.



Authors' Addresses

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

Email: ehalep@ece.upatras.gr

Omar Cherkaoui  
University of Quebec in Montreal  
Montreal,  
Canada

Email: cherkaoui.omar@uqam.ca

Susan Hares  
Huawei  
USA

Email: shares@ndzh.com

Weiming Wang  
Zhejiang Gongshang University  
18 Xuezheng Str., Xiasha University Town  
Hangzhou, 310018  
P.R.China

Phone: +86-571-28877721  
Email: wmwang@zjgsu.edu.cn



Internet Draft  
Intended status: Informational  
Expires: January 6, 2013

S.Hares  
Huawei

July 6, 2012

Analysis of Comparisons between OpenFlow and ForCES  
draft-hares-forces-vs-openflow-00.txt

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), 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 January 6, 2013.

Copyright Notice

Copyright (c) 2012 IETF Trust and the persons identified as the document authors. All rights reserved.

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

Abstract

While both ForCES and OpenFlow follow the basic idea of separations of forwarding plane and control plane in network elements, they are technically different. ForCES specification contains both a modeling language [RFC5812] which allows flexible definition of the Flow tables and flow logic. ForCES flow logic include Logical Functional Blocks (LFBs) connected in flow logic that is described in logic of direct graphs augmented by passage of Metadata and grouping concepts.

OpenFlow's specifications contain a specific instantiation of Flow tables and flow logic which has emerged from the research community theories. OpenFlow's logic varies based on the revision of the specification (OpenFlow-Paper [McKeown2008], OpenFlow Switch Specification 1.0 [OpenFlow1-0], OpenFlow 1.1 [OpenFlow-1.1] Open Configuration 1.0 [OpenFlowConfig-1.0]).

## Table of Contents

1. Introduction.....	3
1.1. ForCES Introduction.....	4
1.2. OpenFlow Introduction.....	4
2. Definitions.....	5
2.1. New Common Configurations.....	6
2.2. Forces Definitions.....	8
2.3. Open Flow Definitions.....	10
3. Comparisons between ForCES and OpenFlow.....	12
3.1. Difference in Historical setting.....	12
3.2. Difference in Goals.....	14
3.3. Difference in Architectural Requirements.....	14
3.3.1. ForCES System Building Blocks.....	15
3.3.2. OpenFlow Building blocks.....	17
3.3.2.1. Match Fields in OFS.....	18
3.3.2.2. Flow Logic - Flow Table and Group tables.....	18
3.3.3. ForCES FE types.....	22
ForCES and OpenFlow FEs can operate either new switching entities or integrated with existing processing as a hybrid. In OFS-1.2, the Ships-in-the-Night (SIN) mode divides existing ports into groups controlled by specific ports (see figure x) or VLANs (figure-x).....	22
3.3.4. ForCES Pre-Association.....	23
3.3.5. Architectural requirements.....	25
3.3.6. ForCES versus OpenFlow - A Central Controller.....	33
3.4. Difference in Forwarding Model.....	33
3.4.1. Looping.....	34
3.4.2. Handling unicast and multicast.....	35
3.5. Difference in Logical Forwarding Block Libraries.....	36

3.6. Difference in Protocol Interface.....	36
3.6.1 Secure Transport.....	37
4. Use of ForCES and OFS in Applications.....	41
5. The use of ForCES or OpenFlow in S(D)N or CSO/SOP.....	41
6. Security Considerations.....	42
7. IANA Considerations.....	42
8. Conclusions.....	42
9. References.....	43
9.1. Normative References.....	43
9.2. Informative References.....	43
10. Acknowledgments.....	44

## 1. Introduction

This document analyzes the differences between OpenFlow and ForCES technically from the aspects of goals, architecture, forwarding models, forwarding policy, control plane interaction, configuration of nodes, and applications (firewalls, load-balancers, High-availability nodes). This informational document compares OpenFlow and Forces as of March, 2012 seeking to provide clarity for other discussions of controller/forwarding split, Software Defined Networking, Software Driven Networking, Cloud Service Oriented networking (CSO), and a host of orchestrators for virtualized network devices.

A fellow Engineer provided inspiration for this deeper comparison by saying: "OpenFlow-0 is the Diff-Serv Tspec, OpenFlow-1.0 is Forces--, and OpenFlow 1.1 is Forces++." Jamal Salim suggests Open Flow 1.1 does not have the same functionality[Jamal-01].

While this summary brings the expert listener quickly into heart of the issues, this document examines:

- Is OpenFlow Switch 1.1.0 really "ForCES++", and "is the group table safe ++ logic? What direction does Open Flow Switch 1.2 and 1.3 take us?
- Where does Open Flow's Config fit in the picture?
- How does this help us get to Clouds Service Oriented Networks (CSO) enable by Software defined networks (SDN) or software driven networks (SDN)?

And that, as the saying goes is the "rest of the story" of this draft. Here's hoping the readers of this document will decide and argue with the author to refine the next-generation of hardware devices.

### 1.1. ForCES Introduction

ForCES (Forwarding and Control Element Separation) work in IETF has defined a new environment to build network devices that split the network devices into control plane and forwarding plane into units. For example, a router could be considered a network element (NE) with a control plane running router protocol and a data plane forwarding IP traffic.

The drive to have ForCES NE device split arose from the desire to build hardware forwarding blades out of flexible hardware components. These hardware devices included Network Processors and network specific ASICs.

The ForCES environment defines requirements [RFC3654], goals [RFC3565], architecture and protocol requirements [RFC3654], a controller-forwarder communication protocol [RFC 5810]. ForCES also describes a policy on how to building the forwarding engine out of a set of logical functional blocks (LFBs) which are connected as a directed graph [RFC5812]. ForCES allows many different Forwarding Engines (FE) to linked to Controller Engines (CE) via the protocols. ForCES provides a modeling language [RFC-5812] to describe these FE devices so that controllers can load control the devices, load forwarding tables, and keep track of statistics. ForCES RFCs also define how the ForCES protocol runs over SCTP [RFC5811].

### 1.2. OpenFlow Introduction

OpenFlow[McKeown2001, p. 1]] arose out of the frustration that network research projects felt at not being able to experiment with new protocols on large-scale networks. Experimentation on research networks did not have a large enough scale to provide a reasonable test-bed for new research ideas for the Internet. Pure commercial networks would not allow experimental protocols, and commercial router vendors took 3-5 years to create a new protocol features. The OpenFlow researchers suggested an alternative to allow the research to creating a slice out of commercial network to try out new ideas for network.

OpenFlow's initial paper grew into OpenFlow Switch Specification versions 1.0 [OFS-1.0], 1.1 [OFS-1.1], 1.2 [OFS-1.2], 1.3 [OFS-1.3], and (likely) 1.4 [OFS-1.4]. Additionally a Config and Management Protocol version 1.0 [OFC-1.0] and version 1.1 [OFC-1.1], and a set of papers and application notes on implementations. A hybrid Specification [OFHy-1.0] suggests how Open Flow may be combined with existing network OpenFlow switches which mix existing

network devices (routers/switches) with OpenFlow controlled switches in either a Ships-in-the Night (SIN) or a hybrid model

OpenFlow's host of specifications and ForCES flexible reprogramming of the network element architecture can be considered the first wave of Software-Defined Networking (SDfN) where software can alter how the forwarding engine's logic operates. This work arises out of the GENI research [GENI][McKeown2008]. The current industry discussion regarding Software-Defined Networking (SDfN) or Software-Driven Networking (SDrN), Network Virtualized Overlays [NVO3], Service-Oriented Protocols (SoP)[SoP] or network orchestrators of Cloud Service Oriented Network (CSO)forwarding [CSO-Arch] have a great deal of confusions as they apply the terms to ForCES and OpenFlow. Rather than carefully define each of these terms explicitly at the outset, we will give brief expansions of the abbreviations and return to the definitions later in the draft after examining the FORCES draft.

This document will examine ForCES and OpenFlow goals, architecture, forwarding conceptual models, Controller-forwarder communication mechanisms and protocols, the policies in the loading of forwarding state, configurations of nodes, and sanity checking of the forwarding.

These basic concepts will be then examined in terms of specific implementations (switch, hybrid router/switch, wireless, load-balancer, firewall) as described by ForCES and OpenFlow reports.

Finally, the document will return to defining S[\*]N, SOP, and Cloud-Oriented Services (CSO).

## 2. Definitions

Definitions used in this document

The key words "MUST", "MUST NOT", "REQUIRED", "SHALL", "SHALL NOT", "SHOULD", "SHOULD NOT", "RECOMMENDED", "MAY", and "OPTIONAL" in this document are to be interpreted as described in RFC-2119 [RFC2119]

The following RFC2119 definitions used in this document are:

The key words "MUST", "MUST NOT", "REQUIRED", "SHALL", "SHALL NOT", "SHOULD", "SHOULD NOT", "RECOMMENDED", "MAY", and "OPTIONAL" in this document are to be interpreted as described in RFC-2119 [RFC2119].

ForCES definitions relevant to this documents discussion are taken from [RFC3654][RFC3746][RFC5810] as noted below. The quoted italicized definitions come from the ForCES RFC, and the non-quoted text applies ForCES RFC text to this document.

## 2.1. New Common Configurations

**Controlling Entity (CE):** is defined as an entity which remote controls the forwarding engine. This Entity can be either a ForCES CE or a Open Flow Controller.

**Controlling Entity Manager (CE-MGR):** This documents loosens the ForCES CE-Manager definition to allow Open Flow and ForCES to be compared. This document defines the CE manager as a logical entity (distributed or located in physical or virtual device) which controls which controllers attach to which logical Forwarding Entities. The Controllers can be in the same physical switch/device in the control plane or other logical software. A CE-Mgr may also be within a VM hypervisor, a VM hypervisor manager, or other virtual software. The CE-Mgr logical function may be distributed across many CE as a defined function. This definitional Allows both ForCES CE-Mgrs and Open Flow Controller collaboration/management via coordinated remote configuration of OF Capable Switches.

**Controlled Router-Switch (CRSW):** A Controlled Switch is a network entity performing switching capability that is controlled by remotely by either the ForCES protocol (FP) or the Open Flow Protocol (OFF). This switch can perform IP routing, MPLS switching, Trill Switching or Layer 2 Switching.

**Forwarding Entity (FE):** is defined as an entity which forwarding packets or frames under the control of the CE. This entity can be an ForCES FE (F-FE) or an Open Flow Capable Switch (OF-CS). An Open Flow Capable switch can either be a hybrid switch or a Open-Flow Only switch.

**FE Manager (FE-MGR):** The FE-Manager controls the FEs assignments. This document defines FE-Manager's logical entity may be a logical software process residing within local switch/device in the control plane or management plane. The FE-Mgr can also within a VM hypervisor, or a VM hypervisor manager, or other virtual software. The FE-Mgr can be a remote service managing the forwarding engine. The Open Flow Configuration [OFC-1.0] Configuration Service point with its logical configuration function may also have a FE-MGR function. This FE-Mgr capability is an capability outside the [OFC-1.0] specification.



Pre-Association Phase (Pre-A): This document defines a Pre-Association Phase (Pre-A) as the period during which a CE-Management (Forces CE-Mgr or OF controller groups) and FE-Managers (Forces FE-MGR (F-FE-MGR) or OF-CS management) determines which Controlling entity (CE) controls which Forwarding Entity (FE).

## 2.2. Forces Definitions

Force Forwarding Element (F-FE) - "A logical entity that implements the ForCES Protocol. FEs use the underlying hardware to provide per-packet processing and handling as directed by a CE via the ForCES Protocol." [RFC3654] ForCES forwarding FE supports forwarding rules insertion.

ForCES Control Element (F-CE) - "A logical entity that implements the ForCES Protocol and uses it to instruct one or more FEs on how to process packets. CEs handle functionality such as the execution of control and signaling protocols." [RFC3654] The ForCES CE controller may be located within the same hardware box on a different blade or across an Ethernet connection, or across a L3 Link (if security used).

ForCES Network Element (F-NE)- "An entity composed of one or more CEs and one or more FEs. To entities outside a NE, the NE represents a single point of management. An NE usually hides its internal organization from external entities and represents a single point of management to entities outside the NE." [RFC3654] The NE's single point of management can be at the IP layer, the Ethernet layer, and at a virtual layer. In this document, the network element is examined as being the set of network functions in the hardware that collaborates to act like a switch. This less strict definition allows ForCES to be compared with the Open Flow work.

ForCES Pre-Association Phase (F-Pre-A): ForCES defines the Pre-Association Phase (F-Pre-A) as "the period of time during which a FE Manager(see below)and a CE Manager (see below) are determining which FE is a part of the network element" [RFC3654].

FE Manager(F-FE-Mgr)- ForCES (F-FE-Mgr)is "A logical entity that operates in the Pre-Association Phase and is responsible for determining to which CE(s) a FE should communicate. This process is called CE Discovery and may involve the FE manager learning capabilities of available CEs." [RFC3654]

CE Manager (CE-Mgr) - Forces CE-MGR[F-CE-Mgr] is "A logical entity that operates in the pre-associaation phase and is responsible for determining to which FE(s) a CE should communicate. This process is called FE discovery and may involve the CE manager learning the capabilities of available FEs. The CE manager may use anything from statics configuration to a pre-association phase protocol." [RFC3654]

ForCES Protocol (ForCES-Proto) - "While there may be multiple protocols used within the overall ForCES architecture, the term "ForCES protocol" refers to only the post-association phase protocol." [RFC3654] The ForCES protocol operates between the "Fp reference points" of the ForCES architecture (as shown in figure 1) [RFC5810]. "Basically, the ForCES protocol works in a master-slave mode in which the FEs are slaves and the CEs are masters." [RFC5810] The location and exact instantiation of the CE logical entities associated with the FE logical entity is flexible. The CE entities could reside on a process on a local switch communicating to other process off the local switch.

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 (including requirements of ForCES TML)" [RFC5810] This layer is defined in 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, multicast, ordering, etc. The ForCES TML specifications are detailed in separate ForCES documents, one for each TML." [RFC5810, p. x]. The ForCES TMLs focused on are STCP [STCP] and SSL[SSL]. TM handles transport of messages [reliable or non-reliable], "congestion control", "multicast", ordering, and other things [RFC5810, p. 14].

LFB (Logical Function Block) - The basic building block that is operated on by the ForCES protocol. The LFB is a well-defined, logically separable functional block that resides in an FE and is controlled by the CE via the ForCES protocol. The LFB may reside at the FE's data path and process packets or may be purely an FE control or configuration entity that is operated on by the CE. Note that the LFB is a functionally accurate abstraction of the FE's processing capabilities, but not a hardware-accurate representation of the FE implementation.

LFB Class and LFB Instance - LFBs are categorized by LFB classes. An LFB instance represents an LFB class (or type) existence. There may be multiple instances of the same LFB class (or type) in an FE. An LFB class is represented by an LFB class ID, and an LFB instance is represented by an LFB instance ID. As a result,

an LFB class ID associated with an LFB instance ID uniquely specifies an LFB existence.

Physical Forwarding Element - The physical element that forwards the packets.

### 2.3. Open Flow Definitions

Open Flow (OF): [McKeown-xx] defines OF as a "way for researchers to run experiments in networks they use every day" [McKeown-2008, p.1].

Open Flow Action [OF-Act]: [OF-1.1.0] defines an OF-Act as an action that may: "forward the packet to a port", or modifies the packet". Actions may be specified in "Open Flow Instruction" [OF-Inst] in Flow Table Entry or "action buckets in Group Table Entry" [OF-1.1.0, p.4].

Open Flow Action Set [OF-ActSet]: [OF-1.1] defines an OF-ActSet as "a set of actions associated with the packet that are accumulated while the packet is processed by each table, and are executed when the packet exits the processing Pipeline." [OF-1.1.0, p. 5].

Open Flow Capable switch [OF-CS]: [One or more physical or virtual switch device which can act as an operational context for an Open Flow Logical Switch [OF-LS]. A OF-CS hosts an Open Flow Data Path [OF-DP].

Open Flow Configuration and Management Protocol [OFCMP]: [OFC-1.0] states the [OFCMP-1.0] enables the remote configuration of Open Flow Data Path (OF-DP). The OFCMP allows a controller to configure the OF-DP on the Open Flow Logical Switch (OF-LS) "so that the controller can communicate and contro" the OF-LS via Open Flow Protocol (OFP). The OFCMP allows dynamic association of resources with OF-LS in an OF-CS. [OFC-1.0] defines the protocol, but not the resource allocation mechanisms.

Open Flow Configuration Point (OFCPT): [OFC-1.0] defines an OFCPT as "a service" which sends OFCMP messages to a OF-CS with an OF-LS inside.

Open Flow Controller [OF-CTLER]: [McKeown-2008] defines the OF-CTLER as a "controller that adds and deletes Flow entries on behalf of experiments" [McKeown-2008, p. 3].

Open Flow Datapath [OF-DP]: [OFC-1.0] defines OF-DP as an abstraction called Open Flow Logical Switch [OF-LS].

Open Flow Dedicated Switches [OF-DS]: [McKeown-2008] defines OF-DS as a "dumb datapath element that forwards packets between ports as defined by a remote process" [McKeown-2008, p3.] The Open Flow process programs the forwarding engine for this dumb datapath switch.

Open Flow Enable Switches [OF-ES]: [McKeown-2008] defines OF-ES as "commercial switches, routers or access points" enhanced by adding the OF feature

Open Flow Feature [OF-Feature]: Open Flow [McKeown2008] and [OF-1.0.0] defines the OF-Feature as adding the features of an Open Flow Logical Switch [OF-LS]. These features are the Open Flow "Flow Tables", "Secure Channel that connects the switch to the controller", and "the Open Flow Protocol" [McKeown-2008, p. 3][OF-1.0.0, p.2].

Open Flow's Flow Table (OF-FT): [McKeown-2008] defines a Flow table in OF as having "an action with every flow table entry to tell the switch how to process the flow" [McKeown-2008, p. 2]

Open Flow's Flow Table Entry (OF-FTE): [McKeown-2008], [OF-1.0.0], [OF-1.1.1], [OF-1.1.2], and [OF-1.1.3] define the specific of an single entry in a flow table. See Section x.x for a detailed comparison of this entry.

Open Flow Group (OF-G): [OF-1.2] defines an OF group (OF-G) as a list of Open Flow "action buckets" and "a means to choose one or more buckets to apply on a per-packet basis" [OF-1.2, p. 5].

Open Flow Group Table (OF-GT):

Open Flow Logical Switch [OF-LS]: OFC-1.0 defines the OF-LS as an abstraction of the "open flow datapath".

Open Flow Packet (OF-Pkt): [OF-1.1.0] defines OF-Pkt as "ethernet frame including header and payload" [OF-1.1.0, p. 4].

Open Flow Pipeline [OF-PipeLine]: [OF-1.2] defines OF-Pipeline as "a set of linked tables that matching, forwarding, and

Open Flow Port [OF-Port]: [OF-1.2] defines an Open Flow port as a place where packets enter and exit the Open Flow Pipeline.

Open Flow Protocol (OFP): OF 1.0 defines "an open protocol to program the flow table in switches and routers" in which "a controller communicates with a switch"[McKeown-2008,p. 2-3].

Open Flow Switch (OFS): [McKeown-2008] defines an Open Flow Switch as a ethernet switch with "at least" the following three functions: "(1) a Flow Table", "(2) "a secure channel that connects the" controller with the switch over which the open flow protocol runs, and (3) an "open flow protocol" [McKeown-2008,p 2.][OF-1.0.0,p.2].

[OF-1.1.0] defines an OFS as "one or more flow tables and a group table which perform packet look-ups and forwarding", and an open flow channel to an external controller"[OF-1.1.0,p.3]. The external controller controls the switch via the Open Flow protocol (OF-Proto)[OF-1.1.0, p.3]. [OF-1.3.0] adds that the "switch communicates with the controller, and the controller controls the switch"[OF-1.3.0, Section 2, paragraph 1.]

### 3. Comparisons between ForCES and OpenFlow

ForCES and OpenFlow are very similar in the following aspects:

- o Both ForCES and OpenFlow are efforts to separate control plane from forwarding plane;
- o Both ForCES and OpenFlow protocols standardize information exchange between the control and forwarding plane.

Although both ForCES and OpenFlow can be considered as the solutions for forwarding and control plane separation, they are different in many aspects. This section compares them in their history, goals, architecture, forwarding model and protocol interface.

#### 3.1. Difference in Historical setting

ForCES work began during the 1995-2000 timeframe with the development of netlink [RFC3549]. The linux netlink began its linux driver history as first a "character device /dev/netlink for Linux kernel 1.3.31" but was superceded by "Alexey Kunzetsov's socket-based af\_netlink.c in Linux v 2.1.15" [Englehardt-2010]. The rtnetlink brought configuration and router queries to links. The netlink socket allowed messages between kernel and user space regarding routes, firewalls, and monitoring.

The historical context of the 1995-2002 timeframe saw the initial growth of the US Internet and spread into non-US sites. The historical changes included changes that began the split of tight binding of control plane to data plane. Forwarding plane elements (ASICs, TCAMs, Network processors) and backplanes began the growth of non-stop forwarding with high-availability. ForCES notes that the data processors have "forwarding tables", "per-flow Qos Tables and access control lists" [RFC365]. ForCES had control processors were general-purpose processors that support route calculations.

ForCES began in quasi-commercial realm of linux development where linux developers used routing software with netlink to build early Internet networks. Alexey's early work was deployed in Russian and European networks to turn linux boxes into Routers. By early 2000, this work had migrated to router boxes seeking to harden routers to provide non-stop forwarding. Netlink implementations were provided with many commercial OEM standards for switches and routers.

ForCES work came out of the desire to expand the basic netlink protocol into an architecture that allow quick modeling of new forwarding hardware and an extensible control-plane to forwarding plane communication. Early discussions in ForCES look to allow coordination of multiple control planes as well as control plane to forwarding plane functionality. However, the IETF decision was to restrict the first versions of ForCES to architecture, CE-FE communication and FE modeling.

OpenFlow arises out of the academic's communities realization that the hardening of commercial of network infrastructure [1995-2006] to support businesses, caused a "reluctance to experiment with production traffic"[McKeown-2008, p. 1]. The GENI (Global Environments for Network Research) [2006-2007] suggested that: a) Internet's infrastructure faced "serious problems" in "security, reliability, manageability, and evolvability" and "possible solutions" existed in research, but there were "severe experimental barriers" to test new ideas in wide-spread deployments [GENI-2007, p. vii]. A new US research network would allow slices of routers to be used for researcher's experiments in network protocols. McKeown and colleagues work examined how these experiments could be extended to run [McKeown-2008] on local campuses. McKeown and colleagues examined persuading commercial routers to provide an open interface for experimentation or using existing open source solutions (linux, XORP[XORP-2008]). Their conclusion was that "commercial solutions are too closed", and "research solutions have insufficient

performance or fanout, and are too expensive." [McKeown-2008, p. 2].

### 3.2. Difference in Goals

RFC3654 lists the the architectural goals of ForCES. OpenFlow Switch (OFS) Specification version 1.0.0 [OFS-1.0.0] and OFS version 1.1.0 [OF-1.1.0] refer to [McKeown-2008] as the basis of these specifications. This document's goal comparison compares the four goals [McKeown-2008] sets against [RFC3654].

The goal ForCES is to define the "architecture", "architectural modeling" and protocols to "logically separate control and data forwarding plans of an IP (IPv4, IPv6, etc.) networking device" [RFC3654, p. 1]. ForCES network device (aka. network element (NE)) can be a router, IP switch, firewall, switch. ForCES redefines network elements to be logical relationships placed on physical devices.

McKeown et al. state the goals of the OpenFlow approach was to find "high-performance and lost-cost implementations" of new network algorithms, capable of being used in "broad range of research", adaptable to commercial "close platforms", and able to "isolate experimental traffic from production traffic [McKeown-2008, p. 2].

Difference in goals underscores the original commercial focus of ForCES and the experimental focus of OpenFlow.

### 3.3. Difference in Architectural Requirements

Architecture sets the building blocks for system, and architectural requirements sets rules for interconnecting the building blocks.

Building blocks for ForCES include the CE(s), FE(s), ForCES protocol (CE to/from FE), FE-Manager, CE-Manager, and logical input flows. Within the FEs there are Logical Forwarding Blocks connected together in a directed graph. The flow processing passes along input port, (modified)frame, metadata (which may include actions). The flow stream may be output to interfaces (logical or physical).

Building blocks for OpenFlow include Controllers (~CEs) and Forwarding Units (~FEs) with OpenFlow processing. OpenFlow logic is designed in terms of Flow Processing controlled by Flow Tables [McKeown-2008][OFS-1.0][OFS-1.1], and Group tables [OFS-1.1] which



operate on the modified frame, metadata, or group of actions via actions or instructions (a group of actions and forwarding commands).

Both flow streams Flow processing may cause the flow to multiple into several streams or combine multiple streams into one.

### 3.3.1. ForCES System Building Blocks

The building blocks within the ForCES architecture are the CEs (controller elements), FEs (forwarding elements), and an interconnect protocol between CE(s) and FE(s). ForCES also recognized the logical functions of a FE-Manager and a CE-Manager. Figure 1 shows a diagram from RFC5810 that details interaction between all these components.

The ForCES CE controls switching, signaling, routing, and management protocols. Each CE is a logical unit which may be located within the same box, different boxes, or across the network. ForCES architecture [RFC3746] allows CEs to control forwarding in multiple FEs.

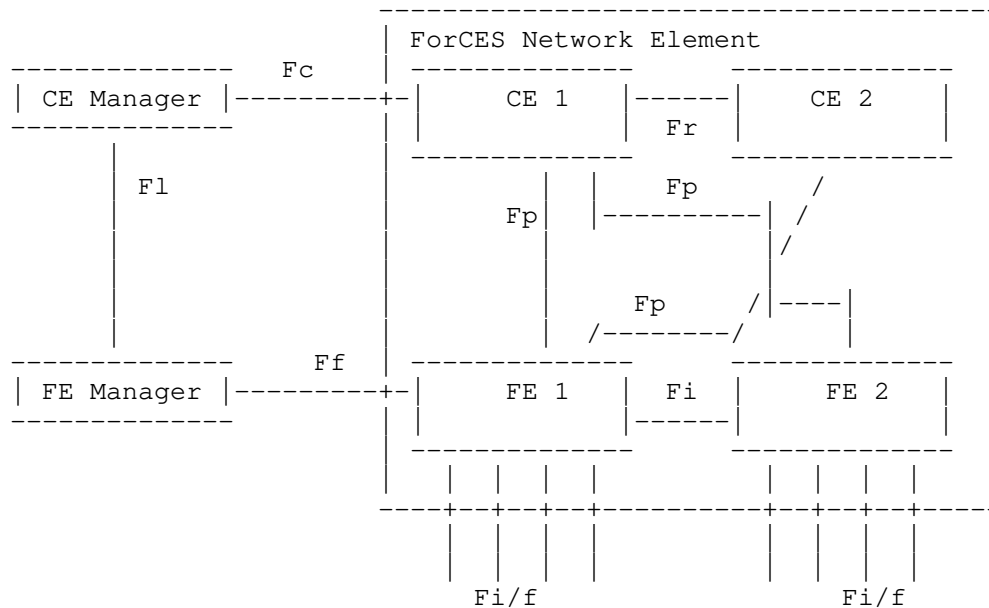
ForCES defines logical Forwarding Elements (FEs) that reside on a variety of physical forwarding elements (PFE) such as a "single blade (PFE)", partition within blade, or multiple PFEs in a single box, or among multiple boxes [RFC3746, p. 2]. The ForCES logical FEs could also be run within Virtual Machines (VMs) within a single box or a set of boxes or a cloud. A single FE may be connected to multiple CEs providing strong redundancy. FE internal processing is described in terms of Logical Forwarding Blocks (LFBs) connected together in a directed graph that "receive, process, modify and transmit packets along with metadata"[RFC5810, p. 6]. The FE model determines the LFBs, the topological description, the operational characteristics, and the configuration parameters of each FE.

The Forces Logical Forwarding Block (LFBs) Library [FORCES-LFB] provides the class descriptions for Ethernet, IP Packet Validation, IP Forwarding LFBs, and Redirection, MetaData, and Scheduling. Forces-LFB document demonstrates how these logical blocks can be placed within a machine to support IP Forwarding (IPv4/IPv6) for unicast & multicast and ARP processing [Forces-LFB, p. 17].

ForCES architecture [RFC3746] allows CEs to control forwarding in multiple FEs. ForCES also recognized the logical functions of a FE-Manager and a CE-Manager. The FE manager determines the CE(s)

each FE should communicate with. The CE manager determines which FEs each CE should communicate with. The ForCES defines the FE-Manager and CE-Manager to operate in a "pre-association" phase of the communication to set-up the FORCES communication path. Similarities between the functions of the CE-Managers and FE managers of ForCES and modern hypervisors may come from the creative interplay of early open source communities [1995-2005]. Applications directly interacting with ForCES components (CEs or CE-Managers or FE-Manager) could be described as interactions with the CEs or CE-Managers.

Figure 1 shows ForCES Architectural Diagram [RFC5810].

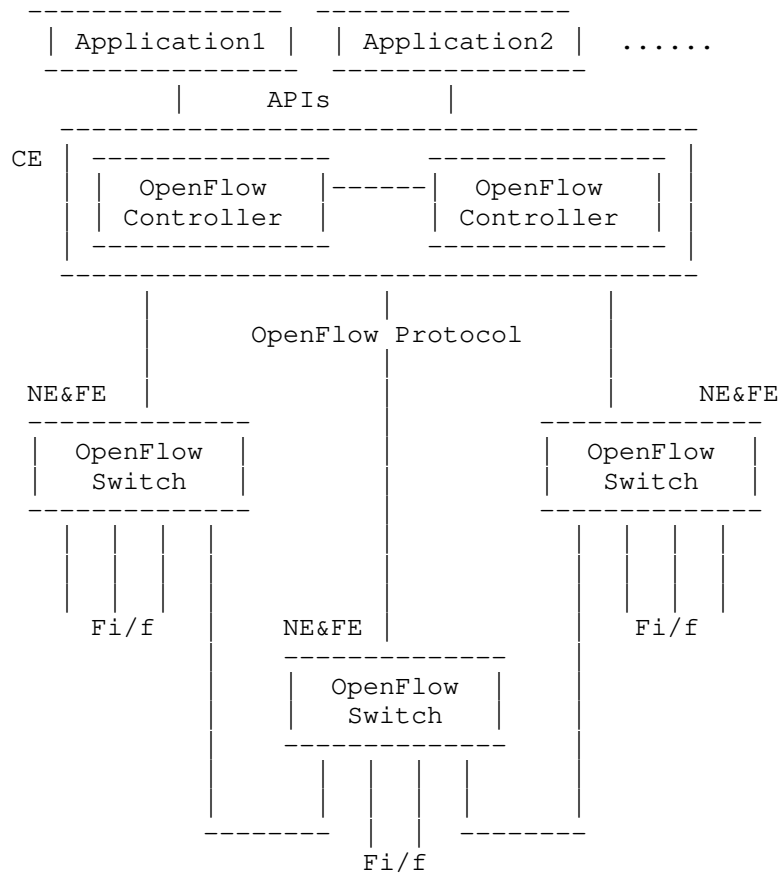


Fp: CE-FE interface  
 Fi: FE-FE interface  
 Fr: CE-CE interface  
 Fc: Interface between the CE manager and a CE  
 Ff: Interface between the FE manager and an FE  
 Fl: Interface between the CE manager and the FE manager  
 Fi/f: FE external interface

Figure 1: ForCES Architectural Diagram [RFC5810]

### 3.3.2. OpenFlow Building blocks

OpenFlow architecture consists of set of OpenFlow Switches with a Flow Table, a Secure Channel between controller and switch, and an Open flow protocol. OpenFlow switches can either be only controlled by OpenFlow, enabled by Open Flow [OFS 1.0] (shared), or hybrid Switches (OFS 1.2).



Fi/f: FE external interface

Figure 2: OpenFlow Architectural Diagram  
by Using the terms NES, FEs, CEs

The Flow table provides entries on how to process a flow whose header fields match a pattern in the header field [OFS-1.0.0] or

a set of meta data generated from pipeline processing of a header [OFS-1.1.0, figure 4] [OFS-1.3].

The matching of a packet in OFS-1.0.0 based on first exact match of header and/or meta data, and secondly wild-card entries. The wild-card entries contain a priority field to order the process of matching. For example, a priority of "1" will be the first wild card processed.

### 3.3.2.1. Match Fields in OFS

The match field has been expanding as the ONF specifications evolve - from a 10-tuple [McKowen-2008], to 12-tuple [OFS-1.0], and to a OFS-1.1.0 a 15-tuple, to 39-tuple in OFS-1.3 [OFS-1.3] (14 required and 25 optional).

The original 10-tuple includes ingress port, VLAN ID, Ethernet source address, destination address and type, the IPv4 source/destination address, and IP protocol, TCP/UDP source & destination port. The 12-tuple adds VLAN priority, and IP Tos bits. The 15-tuple adds: metadata, MPLS label, MPLS traffic class. The TCP/UDP source & destination port have redefined for ICMP packets to have instead the ICMP Type and ICMP code. [OFS-1.3-pre] required matches include the 10-tuple plus IPv6 source and destination addresses and UDP source and destination ports.

### 3.3.2.2. Flow Logic - Flow Table and Group tables

The flow table operation and data structures vary based on the version of OpenFlow Switch specification.

OFS in versions [McKeown-2008] and [OFS-1.0.0] operate on logic in flow tables which are executed in ascending order. Each Flow Table ID must be greater than the current Flow table id. [OFS-1.1.0] [OFS-1.3] flow logic operates on flow tables and group tables which allow jumps based on "Goto table" logic or combinations of flows.

```

|=====|      |=====|      |=====|
|Flow table 0 |---.|Flow Table 1 | _ -. |Flow Table n |
|=====|      |=====|      |=====|

```

Figure 3: Flow Table [OFS-0.9][OFS-1.0]

All OFS Flow tables match on data and perform actions [OFS-0.8][OFS-0.9][OFS-1.0][OFS-1.1][OFS-1.2][OFS-1.3]. Later versions [OFS-1.1.0][OFS-1.2][OFS-1.3] use instructions to immediately perform actions or to queue specific actions for later processing. These same later versions also allow metadata to be stored to be passed along to additional processing.

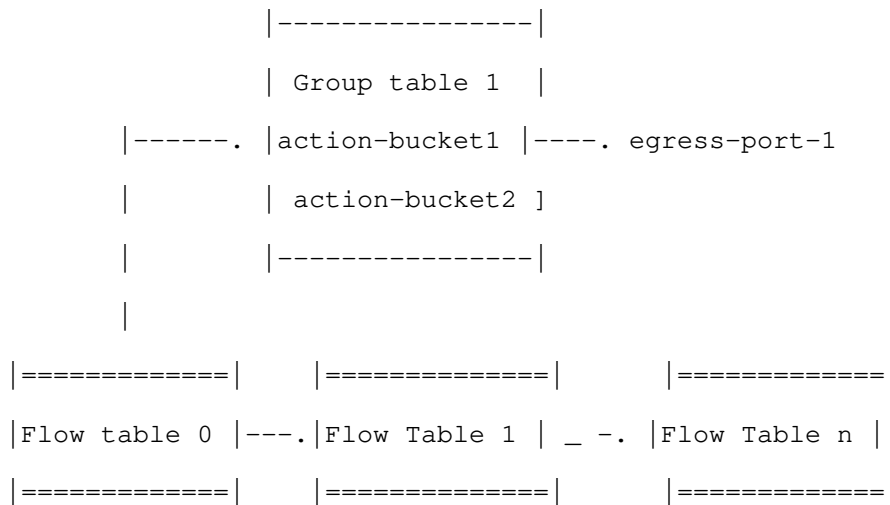


Figure 3: Flow Table [OFS-1.1]

#### 1) Action Specifics

[McKeown-2008] has three actions in the flow tables: forwarding of a matched packet to a specific port or ports, sending the packet to the controller, or dropping the packet. This simply processing is why some engineers suggest that OFS-2008 is similar to the RSVP T-Spec [RFC2870].

[OFS 1.0.0] actions direct switch processing to forward packet, drop packet, enqueue packet (optional), and modify-field in packet (optional). Forwarding packets can be sent to all ports, the controller, local switches forwarding stack, send out input port, and specific ports after performing table actions & send out specified port, send via normal (L2/L3/VLAN) processing, and

flood (via minimum spanning tree) ports. This causes some engineers to consider OFS 1.0 equivalent to be Forces--.

[OFS-1.1.0] uses instructions within each flow entry to determine how a packet and associated data is processed. These associated data includes: ingress port packet came in on, the generated meta-data and an action set. The action set is a set of commands to execute prior to sending the packet out. The [OFS-1.1.0] instructions are: "Apply-Actions", "Clear-Actions", "Write-Actions", "Write-metadata", "Goto Table" [OFS-1.1.0, p. 14]. Actions are either applied immediately with apply action command, or stored (via an Write-Action) in action sets for later processing in the action-set via a Write-Action. The existing actions can be cleared with a "Clear-Action". [OFS-1.1.0] actions are output packet, set queue, drop packet, process via group table, push/pop tags, and set-field. [OFS-1.1.0] action sets include single entries for any of the following: copy TTL inward in packet, pop/push actions to packet, copy TTL outwards, decrement TTL, set fields in packet, apply QoS Actions (e.g. set queue), and apply group actions, and output packet.

## 2) Flow Logic Encoding

[OFS-1.0.0] encodes the ForCES LFB in table sequences. The LFB directed graph of ForCES modeling is encoded in sequences of Flow Table. OFS 1.0 specifies that the Ethernet header (similar to ForCES Ethernet II) is the basic frame for all input. A fixed processing ARP, IPv4, TCP/UDP, and ICMP packets are specified based matches beyond the Ethernet header. The Forces LFB library provides building blocks for matches beyond the Ethernet to ARP, IPv4 and IPv6 packets, and Meta data. The OFS-1.0.0 does not have Metadata.

[OFS-1.1.0] match of a flow goes against specific table 15-tuple header. If the frame/packet matches, the flow table can alter the packet (via immediate actions), add metadata (for later handling), set an actions in action set, pass the processing to a specific table (Flow Table or Group Table), or pass the processing to the next table in the sequence. The information passed on to the next processing is [ingress port, (post-modification) frame/packet, metadata, and action set.

[OFS-1.1.0] allows processing between tables to carry the ingress port, the packet, generated metadata, and an action set. An action set is a set of commands to execute prior to sending the packet out. [OFS-1.1.0] uses Flow table with instruction. The instructions can be which can be "Apply-Actions", "Clear-

Actions", "Write-Actions",  
"Write-metadata", "Goto Table" [OFS-1.1.0, p. 14]. Actions are either applied immediately with apply action command, or stored (via an Write-Action) for later processing in the action-set via a Write-Action. The existing actions can be cleared with a Clear-Action.

[OFS-1.1.0] supports two types of tables: flow tables, and group tables. The group table structure has a group identifier, group type, counters, and an order list action buckets. Action buckets contain a set of actions with parameters. If the group table has "zero" action buckets, then no processing occurs.

[OFS-1.1.0] group type field specifies how the action buckets operate on the packet. The group can be "all", "select", "indirect", and "fast-failover" [p.7]. The "all" bucket provides multicast and/or broadcast support execute all buckets. The packet is effectively cloned and sent out. The select, indirect, and fast-failover execute one bucket. In select, the switch determines which port via internal algorithm (e.g. round-robin). In "indirect", the group table logic selects the bucket. The "fast-failover", the first live bucket is chosen. The single-bucket choses may restrict flows if the specified buckets and/or output ports are down.

This new sequential logic is the basis of some engineers comment that OpenFlow 1.1 is ForCES++.

ForCES people indicate that that the group table logic plus "Go to" logic is simply a LFB model of a specific type. [Haleplidis-2012] demonstrates on the LFB library concept can be used to capture all of the OFS-1.1.0 specification.

## 3.3.3. ForCES FE types

ForCES and OpenFlow FEs can operate either new switching entities or integrated with existing processing as a hybrid. In OFS-1.2, the Ships-in-the-Night (SIN) mode divides existing ports into groups controlled by specific ports (see figure x) or VLANs (figure-x)

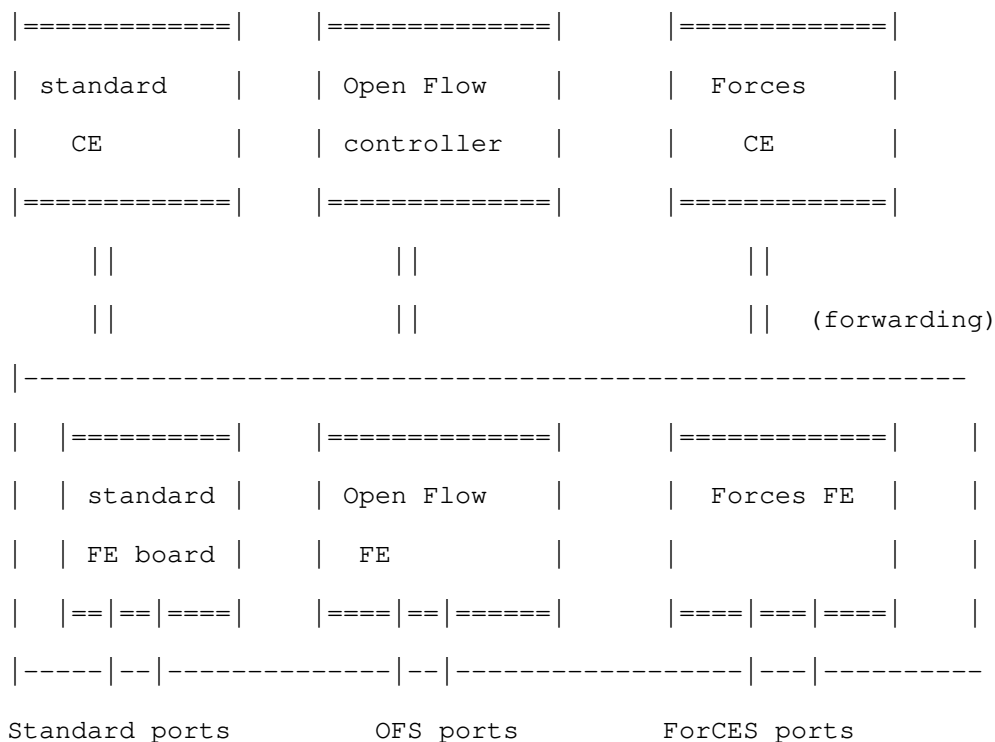


Figure x - Hybrid mode per port



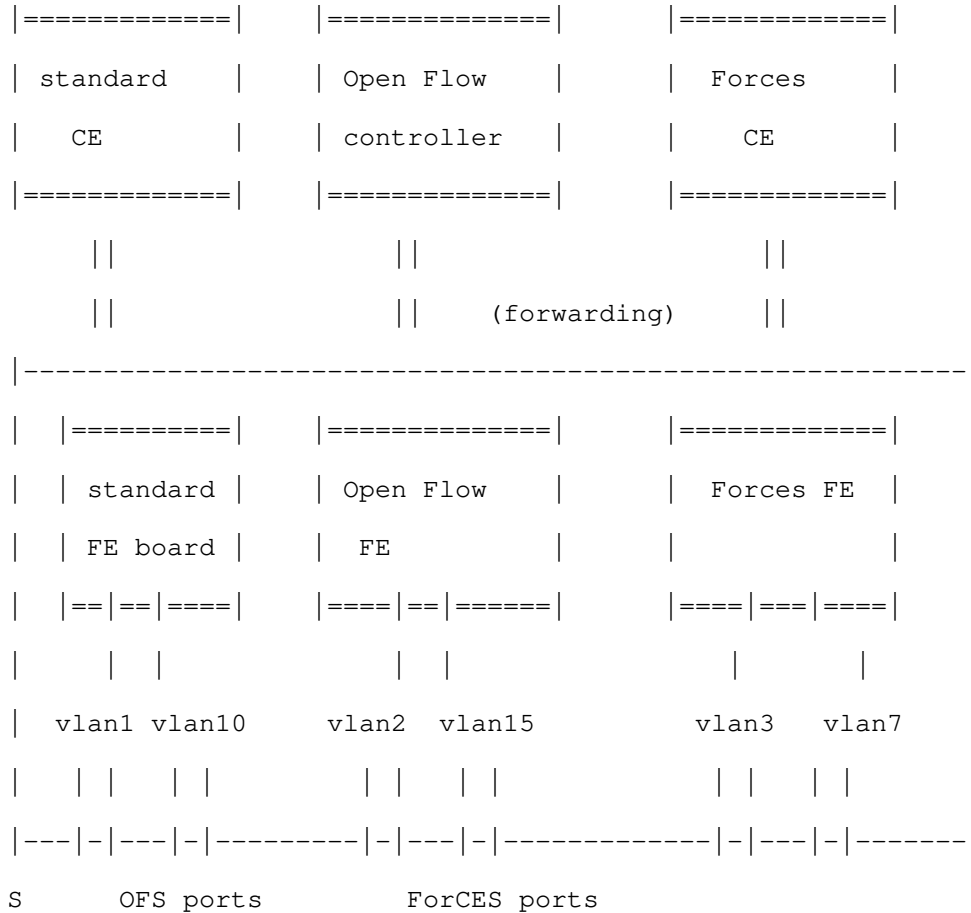


Figure x - Hybrid mode per port

### 3.3.4. ForCES Pre-Association

Neither the ForCES protocol nor the OFS protocols [McKeown-2008/OFS-0, OFS-1.0.0, OFS-1.1.1] specify how the CEs/controllers and FEs/forwarding switch meet.

Do CEs go to the FEs meeting place and CEs pick up the FE that delights their forwarding fancy? How do they found out where

eligible FEs are meeting? Do FEs have choice on which CEs select them, and if so what are the criteria?

The forming of intimate relationships between CEs and FEs remains to the readers of the specification as mysterious as the pre-association stages of human group dating or clique forming.

ForCES specifications specifically call this phase a pre-association phase. The ForCES architecture names the entity that coordinates the forming of associations (like a Jewish match-maker) for CEs and for FEs. The CE-Manager determines which FEs each CE should talk to. The FE-Manager determines which CEs each FE should talk to. Only when the associations between each CE and its FEs, and each FE and its CEs are complete, does the system complete pass from pre-association phase to association phase.

It is assumed that some protocol interactions within the logical ForCES network entity (or entities) determine how CEs will coordinate their work. However, the IETF work specifically denoted this CE-CE coordination work as a second phase of work.

OpenFlow Switch specifications ([McKeown-200][OFS-0.8][OFS-0.9][OFS- 1.0] OFS-1.1] ignore the concept of this dynamic meeting processing.

Either OFS specification missed this concept. Perhaps the OFS specifications assumed a static configuration as part of a boot process of the hybrid switch will set-up some basics. It is possible that the lack specification may come from the sponsors of the specification wanting proprietary pre-association interactions. If so, this provides an interesting line of demarcation between standards and OFS standard.

In any case, this oddity from the OFS proponents leaves one to ask "What is the rest of the story on the pre-association phase?"

## 3.3.5. Architectural requirements

RFC3654 specifies 15 architectural requirements. Table 15 provides summary of this requirements and possible OFS (McKeown-2008/OFS 0, OFS 1.0, OFS 1.1).

ForCES Architectural Requirement	OpenFlow Switch Architecture
-----	-----
1. CE/FEs connect via variety of communicate interconnect technologies. [RFC3654, p. 5]	Controllers/switch over a secure connection [McKeown-2008][OFS-1.0][OFS-1.1]
FE can use different technology than CE/FE topologies.	not specified
2. "FEs MUST support a minimal set of capabilities necessary for establishing network connectivity (e.g. interface discovery, port up/down functions.) Beyond this minimal set, the ForCES architecture MUST NOT restrict the types of numbers of capabilities that FEs may contain. [RFC3654, p.5]	[McKeown-2008][OFS-1.0][OFS-1.1] specify a set of required actions, instructions, Flow actions, and an implied set of port functions(e.g.interface discovery, port up/down). These OFS specifications also declare some set of features optional.
3. "Packets MUST be able to arrive at the NE by one FE and leave the NE via different FE." [RFC3654, p.5]	[OFS-1.0][OFS-1.1]specifies in ofp_port the OFPP_IN_PORT flag which allows the port to explicitly send it back out the input port [OFS-1.0, p. 18] [OFS-1.1, p. 26]

ForCES Architectural Requirement	OpenFlow Switch Architecture
-----	-----
4. "A NE must support the appearance of a single functional device." (e.g. single IP TTL reduction.) [RFC3654,p. 5]	[OFS-1.0][OFS-1.1] describes the devices as an individual switch, and provides the capability to reduce TTL [OFS-1.1,section 4.7, p. 12]  such as push/pop of VLAN IDs and/or MPLS headers [OFS-1.1, p. 12]
4b. However, external entities (e.g. FE managers and CE managers) MAY have direct Access to individual ForCES protocol elements for providing information to transition [RFC3654, p. 5]	4b. No pre-association logic has been defined.
5."The architecture MUST provide a way to prevent unauthorized FORCES protocol elements from joining an NE." [RFC3654, p. 5]	Beyond a secure channel between some "controller entity and switch" no prevention of unauthorized access has been encoded.
6. A FE must be able to asynchronously inform the CE of a failure or increase/decrease in available asynchronous, resources or capabilities on the FE.	[OFS-1.0][OFS-1.1] have "three message types: controller-to-switch, symmetric [OFS-1.0, p. 10] [OFS-1.1, p. 16]. Asynchronous messages

## ForCES Architectural Requirement

## OpenFlow Switch Architecture

-----

-----

Switches send a controller (CE) Messages with a received packet (packet-in), notification of a removed flow table entry (flow-removed), changes in port status (port-status), and error conditions (error) [OFS-1.0, pp-10-12][OFS-1.1, p. 16-17]

"Thus, the FE MUST support error monitoring and reporting" (e.g. "number of physical ports Or "memory changes"). [RFC3654,p. 5]

The change in port status is covered, but memory status is not covered

7. "The Architecture MUST support mechanisms for CE redundancy or CE failover.

[McKeown-2008], [OFS-1.0] [OFS-1.1] do not specifically provide CE Redundancy or CE failover.

1. This includes the ability for CE and FEs to determine when there is a loss of association between them, ability to restore the association and efficient state (re)synchronization mechanisms.

The OFS protocol supports echo request/reply message [OFS-1.0, p. 41] [OFS-1.1, p. 55-56].

The OFS-1.1 provides a barrier message that provides synchronization [OFS-1.1, p. 50].

## ForCES Architectural Requirement

-----

7. (CE redundancy - continued).

2. This also includes the ability to preset the

## OpenFlow Switch Architecture

-----

The OFS-1.1 protocol supports query by the controller of the switch features, capabilities, configuration, flow table configuration, flow table entries, group table entries, port configuration (pp. 36-42). FS-1.1 also provides Statistics on description of Switch (OFPST\_DESC), Flow table status (OFPST\_FLOW), aggregate flow statistics (OFPST\_AGGREGATE), port statistics (OFPST\_PORT), queue statistics (OFPST\_QUEUE), group statistics (OFPST\_GROUP), and experimenter extension (OFPST\_EXPERIMENTER) (p.43-49).

OFS-1.1 states:

"In the case the switch loses

actions an FE will take in reaction to loss of association to its CE (e.g., whether the FE will continue to forward packets or whether it will halt operations." [RFC3654, p. 6.]

contact with the current controller, as a result of an echo request timeout, TLS session timeout, or other disconnection, it should attempt to contact one or more backup controllers. The ordering of the backup Controllers is not specified by the protocol."

"The switch should immediately enter either "fail secure mode", or "fail standalone mode" if it loses connection to the controller, depending on the switch implementation and configuration." [OFS-1.1, p.18]

In "fail secure mode", the FE behavior remains the same except FE drops "packets and messages" destined for CE. In "fail-standalone mode", FE processes all packets acts as a "legacy switch or router." [OFS-1.1, p. 18]

- 
- Flow entries persist over Failure in either fail secure Mode or fail standalone mode.
8. "FES must be able to redirect control packets addressed to their interfaces to the CE. The (FE) MUST also redirect other relevant packets (E.g., such as those with Router Alert Option set) to their CE. The CEs MUST be able to configure the packet redirections information/filters on the FEs.
- [McKeown-2008][OFS-1.0][OFS-1.1] allow packets to forward to Controller for processing.
- [OFS-1.0][OFS-1.1] Flow tables allow packet redirection filters on the FEs with action Forward controller (OFS-1.0, p. 6), (OFS-1.1, p. 13).
- The CEs MUST also be able to create packets and have its FEs deliver them.
- [OFS-1.0][OFS-1.1] allow a CE to FE message (packet-out) to send frames/packets out a specific port [OFS-1.0, p. 10], [OFS-1.1,p.17]
9. "Any proposed ForCES architecture MUST explain how that architecture supports all the router functions as defined in [RFC1812]."
- [OFS-1.0][OFS-1.1] do not consider this requirement. Future OFS work may consider this set of features.
1. Includes: IPv4 Forwarding Options
  2. Should include: IPv6 forwarding options



## ForCES Architectural Requirement

## OpenFlow Switch Architecture

- | ForCES Architectural Requirement  | OpenFlow Switch Architecture   |
|---|--|
| -----   | -----  |
| 10. "In a ForCES NE, the CE(s) MUST be able to learn the topology by which the FEs in the NE are connected."                              | [OFS-1.0][OFS-1.1] do not consider this requirement.   |
| 11. The ForCES NE architecture MUST be capable of supporting (i.e. must scale to) at least hundred of FEs and tens of thousands of ports. | [McKeown-2008], [OFS-1.0][OFS-1.1] do not consider scale of CE/FE  |
| 12. "The ForCES NE architecture MUST allow FEs and CEs to join and leave NEs dynamically."  | [McKeown-2008], [OFS-1.0], [OFS-1.1] do not consider issues relating to active join/leaving of CEs and FEs in communication. |
| 13. "The ForCES architecture MUST support multiple CEs and FEs. However, coordination between CEs is out of scope of ForCES."             | [McKeown-2008], [OFS-1.0], [OFS-1.1] do not directly discuss how multiple CEs will attach to FE. Or FEs attach to a CE.      |
| [Historical note:<br>The restriction of CE coordination was a desired phase 2 work of the ForCES group.]                                  |  |

## ForCES Architectural Requirement

## OpenFlow Switch Architecture

14. For pre-association phase set-up, monitoring, configuration issues, it MAY be useful to use standard management mechanisms for CEs and FEs.

The ForCES architecture and requirements do not preclude this.

In general, for post-association phase, most management tasks SHOULD be done through interactions with the CE. In certain conditions (e.g., CE/FE disconnection), it may be useful to allow management tools (E.g., SNMP) to diagnose and repair problems.

The following guidelines MUST be observed:

1. The ability for a management tool (e.g., SNMP) to be used to read (but not change) the state of FE SHOULD NOT be precluded.
2. IT MUST NOT be possible

[McKeown-2008], [OFS-1.0], [OFS-1.1] do not consider issues relating setting up the links between CEs and FEs. Some "magic" occurs and the CE is talking to a particular FE.

[OFS-1.0][OFS-1.1] give no discussion on other management process (SNMP) outside the [OFC-1.0] using netconf and beep

for management tools  
(e.g., SNMP, etc) to  
change the state of an  
FE in a manner that  
affects overall NE behavior  
without the CE being notified.

### 3.3.6. ForCES versus OpenFlow - A Central Controller

ForCES and OpenFlow seek to split the control plane and the forwarding engine. Both protocols using a secure connection can be used to interact with a central controller. ForCES has spent more time determining how CEs and FEs might find one or more central controllers. OpenFlow Specifications are just beginning to rediscover the need for this work.

Both Forces an OpenFlow can provide the ability of a logically centralized controller to:

- o Collect the network view and make decisions according to control logics (or applications);
- o Interact with forwarding hardware (FE) to install forwarding policy and state,
- o Provide open APIs to users to add new features.

ForCES has considered security issues (such as Denial of Service (DOS)) and the mechanisms for grouping CEs with an FE, or FEs with a CE, Forwarding Models, and Forwarding Libraries.

OFS specifications have focused on defining one simple functionality that can be implemented in specific networks. For example, many discussions point to code deployed in Google.

### 3.4. Difference in Forwarding Model

ForCES and OFS pipeline processing of frames/packets include the basic steps of matching framing, processing frame, and outputting/dropping frame. The processing of a frame occurs in a pipeline of processes where initially processing adds the metadata and actions that subsequent processing will use to create the final packet that will be sent via output port or dropped.

Key ingredients of a good pipeline process are:

- 1) a deterministic logic based that doesn't loop,
- 2) handles both unicast and multicast traffic,
- 3) Flexible matching that can growth with new features,
- 4) Metadata to allow passing of results to subsequent stages,
- 5) Logic that allows some stages to be skipped, and
- 6) Allows for no match (Table Miss).

#### 3.4.1. Looping

In ForCES, [RFC5812] defines the FE (Forwarding Element) model based on an abstraction of Logical Functional Blocks (LFBs). In this model, each FE is composed of multiple LFBs that are interconnected in a directed graph, which is represented by the LFB topology model. The directed graph model prevents the recycle of processing in a loop. Each LFB defines a set of processing on handling frames/packets. For example, typical LFBs include IPv4/IPv6 Longest Prefix Matching, etc. XML is used to describe LFB model formally.

In [OFS-0.8][OFS-0.9][OFS-1.0] the forwarding model has been static defining specific functions for early experimentation of the switch. Loops have been prevent

[OFS-1.0] defines the Flow tables identified by a sequential number [0,1,2\_n]. Processing loops are prevent by defining that a flow table can only transfer to a higher flow table.

[OFS-1.1] provides a Group table to augment the Flow Table logic described above. If a flow matches, instructions in the flow table may direct the packet toward specific table (group table or flow table). This jumping provides skips in sequential process unlike [OFS-1.0]. In addition, the "goto" action allows skips between tables.

### 3.4.2. Handling unicast and multicast

[OFS-0.9][OFS-1.0] do not provide an easy way to provide cloning for multicast. The group table in [OFS-1.1] provides the necessary cloning for multiple outputs of a single packet.

A ForCES IPv4MultiLPB and IPv6MultiLPB could be defined beyond today's ForCES standards. These LFBs would use an LPM to match the multicast address, and generate a list of "L3PortID" metadata to identify a set of ports the cloned packet could be sent out. This metadata could be passed to the EthernetEncap LFB.

### 3.4.3. Flexible matching

All OFS specifications ([OFS-0.8][OFS-0.9][OFS-1.0][OFS1.1][OFS1.3-pre]) seek to match the header data against the flow table's match field. Ranging from a 10-29 possible matches in the header and metadata, the OFS provides flexible matching within the data packet (Ethernet, MPLS, IP, TCP, UDP).

[Heleplidis-Forces-LFB] shows the matching capability in OFS-1.1 can be implemented in ForCES LFBs.

### 3.4.4. Metadata to allow passing of results to subsequent stages,

Both ForCES and OFS ([OFS-1.1][OFS-1.3-pre]) allow metadata to be passed to subsequent spaces.

### 3.4.5. Optionally skipping logic

[OFS-1.1] provides a Group table to augment the Flow Table logic described above. If a flow matches, instructions in the flow table may direct the packet toward specific table (group table or flow table). This jumping provides skips in sequential process unlike [OFS-1.0]. The Group Table concepts provide the ability to group flows for execution, single out a single flow for additional processing, and use port liveness mechanisms for fast-failover. [OFS-1.1,p, 7].

The ForCES directed graph model can also allow the skips in processing by having multiple exits from.

### 3.4.6. Table Miss

A frame may not match any table in the forwarding pipeline.

[OFS-0.9] states "if no matching entry can be found for a packet, the packet will be sent to the controller over the secure channel"[p. 8].

Experience has taught the OpenFlow community this can be problematic.

[OFS-1.0.0-errta] states the following exceptions: "Sending the packet to the controller on table-miss may overload the switch or the controller, and some of those packets may have to be dropped, and this may be an issue in some deployments" [p., 3].

The OFS-1.0.0-errta suggests that the vendor extension may allow the packet to be dropped or forwarded via pipeline. However, due to many application use of table-miss to do topology discovery or watch traffic - this feature is continued.

ForCES does not define a global Table-Miss, but allows the LFB model to define these issues.

### 3.5. Difference in Logical Forwarding Block Libraries

The Open-Flow group is beginning to consider flexible description of the next OFS switches using a modeling language. No modeling language has been approved as yet.

The Force LFB Library [ForCES-LFB-Lib] has been defined and implemented. [Heleplidis-Forces-LFB] shows the modeling language can support OFS-1.1 definitions.

### 3.6. Difference in Protocol Interface

The OFS protocol and the ForCES protocol both use:

- . Secure transport protocols over which they operate (3.6.1)
- . Messages to establish the Controller/CE - FE connections,
- . Messages to Loading of forwarding logic (3.6.3)
- . Messages to Configuration the box (3.6.4),
- . Error handling messages (3.6.5),
- . Liveness protocols (3.6.6), and

- . Sending packets for processing to/from controller (3.6.8).

### 3.6.1 Secure Transport

ForCES defines two layers of protocols: ForCES Protocol Layer (ForCES PL) and ForCES Protocol Transport Mapping Layer (ForCES TML).

ForCES PL defines Protocol between FEs and CEs (Fp Reference Point). ForCES Protocol Transport Mapping Layer (ForCES TML) is defined to transport the PL messages. It is expected that more than one TML will be standardized and interoperability is guaranteed as long as both endpoints support the same TML. [RFC5811] has defined a SCTP-based TML for ForCES.

OpenFlow defines the protocol between controller and OpenFlow switches, i.e. OpenFlow protocol. OFS-1.1 states that the data channel is "usually encrypted using TLS, but may be run over TCP"[OFS-1.1.0,p. 16]

### 3.6.2 Types of Messages

As Table-x shows, ForCES and OFS protocol are remarkably similar. Many OFS authors indicate the influence of the ForCES protocol on the OFS work. Due to the IETF review, ForCES protocol's top level is carefully designed with orthogonal features of association setup, association teardown, config, query, events, packet redirect, and heartbeat.

The OFS protocols [OFS-0.8][OFS-0.9][OFS-1.0][OFS-1.1] provide similar features, but have some overlapping functions. Similarly, the OFS protocol has

- o Hello (initial association),
- o Reading of switch Features (read/response),
- o config of switch and flow pipeline's via Flow tables [OFPT\_FLOW\_MOD], group tables [OFPT\_GROUP\_MOD], ports [OFPT\_PORT\_MOD].
- o Flow Removed [OFPT\_FLOW\_REMOVED] - Flow entry is removed due to either an idle timer or a hard timeout.
- o Query of statistics (OFPT\_STATS\_REQUEST, OFPT\_STATS\_RESPONSE),

- o Packet-OUT- redirect of packet from controller out a port,
- o PACKET-IN - redirect packet inbound port to controller,
- o Echo request/reply - heartbeat from OFS switches.

In addition, the OFS protocol has a Barrier Request/reply message that allows the controller to synchronize message processing. This general (but lose) definitional has allowed experimentation with OFS switches.

Due to IETF review, the similar ForCES protocol has a clear orthogonal set of actions described in terms of execution and transaction models. The CE can set execution flags on sets on transactions (groups of functions). The execution of transactions can be: execute-all-or-none, continue-execute-on-failure, and execute-until failure. A transaction set is must me the ACDity test (atomicity, consistency, isolation, and durability) [RFC5810, section 4.3.1.2]. Transaction sets have an start, middle, and end. The transaction can also signal an abort.

The notification messages for Error and Port status are uniquely specified in the OFS as a notification. In ForCES this is included with the general notification category.

Table-x ForCES vs. OFS messages

Message:	ForCES	OFS-0.9	OFS-1.0	OFS-1.1
=====				
Associate		-----Hello-----		
CE/FE (Req/Rsp)	X	X	X	X
Association		-----Feature Request/Response---		
Stop (Req/Rsp)	X	X	X	X
Query/	X	-----Feature or STATS (Req/Rsp)---		
Query Response	X	X	X	X
Config [Switch]		---Config (non-flow table)-----		
(Req/Rsp)	X	X	X	X



Config (Req/Rsp)	X	-----	FLOW_MOD	-----
	X	X	X	X

Table-x ForCES vs. OFS messages

Message:	ForCES	OFS-0.9	OFS-1.0	OFS-1.1
=====	=====	=====	=====	=====
Heartbeat (Forces)	X	---	Echo-Request/Response	---
/Echo-Request (OFS)	X	X	X	X
Redirect		---	Packet_in & Packet-Out	
	X		X	X
Execution flags	X		----	Barrier-Set-Queue
			X	X
Notification	x		X	X

### 3.6.3 Loading of forwarding logic

ForCES and OFS both use TLVS to add, modify, and delete the flow entry. In addition, ForCES has a concept of "commit" to a set of changes to allow multiple stages of set.

OFS has the concept of modifying or deleting only strictly matching flows (OFPFC\_MODIFY\_STRICT, OFPFC\_DELETE\_STRICT). This is different that the OFS default of modifying all flows with that match (with wildcard).

### 3.6.4 Configuration

ForCES defines changing configuration of the switching Forwarding pipeline within the protocol. OF-Config-1.0 has provided a protocol to use an OpenFlow Configuration Point (logical mode) that can configure one or more OFS via the OpenFlow configuration protocol. The configuration protocol runs on top of TLS or TCP. The configuration protocol sets the following information:

- o Failure standby mode (fail secure or fail standalone)

- o Encryption mode (TLS or not),
- o Queue configurations (min-rate, max-rate, experimenter),
- o Ports (speed, no-receive, no forward, no packet-in, link-down, blocked, life) and optionally (duplex-mode, copper-medium, fiber-medium auto-negotiation, pause, asymmetric-pause),
- o Data path id of switch.

### 3.6.5 Error handling and sanity checking

The error handling indicates errors that occur within the protocol. The Error handling includes message form and action failure. For OFS the action failure includes all interactions such as: hello failure, bad request, bad flow action, bad flow instruction, bad match, cannot modify entry in flow table, cannot modify entry in group table, cannot modify port.

Due to IETF review, the ForCES errors and notifications are define to contain all cases within the protocol. The error processing contains sanity checking.

### 3.6.6 Failure of CE/FE connection

Heart beat messages in both ForCES and OFS insure "liveness" of the CE/FE connection. The ForCES heartbeats are traffic sensitive, and are only sent if no traffic has occurred.

OFS predefines that switches should enter the following based on losing connections with controller: "Fail secure mode" or "fail standalone mode" [OFS-1.1.0,section 5.3]. In Fail secure mode, forwarding continues as previous with the only change that no packets can be uploaded to the processor.

In fail standalone mode, the OSF switch drops into the Ethernet legacy mode [OFPP\_NORMAL].

If the ForCES protocol is supporting the high-availability function, the begins the engage the high-availability statement machine. OpenFlow specifications have not yet described how High-availability will work in Open-Flow.

#### 4. Use of ForCES and OFS in Applications

ForCES and OFS [OFS-1.0][OFS-1.1] have been encoding in a variety of applications. These application include:

- . Firewalls,
- . Loads balancers,
- . Switches
- . High-availability routers,
- . Wireless devices.
- . Table-x ForCES vs. OFS messages

#### 5. The use of ForCES or OpenFlow in S(D)N or CSO/SOP

This section will contain a summary of the common capabilities of ForCES and OpenFlow in environments of centralized controllers, distributed controllers, and hybrid (centralized/distributed control) suggested by open flow.

##### 5.1 - Centralized controller logic

ForCES and OFS have been designed for centralized controller logic. ForCES has considered the pre-association and association phase of the CE-FE relationship with all the timing issues. The execution and transaction model provide a strongly reviewed model to provide roll-forward and roll-back of transactions. The high-availability drafts for ForCES provide a clear case on how to keep high-availability of forwarding and CE processing while distributing the flows.

ForCES has a clear body of work developed over years of implementation experience.

OFS specifications do not deal with how controllers find FEs. However, numerous companies are developing centralized controllers. The standardization efforts for Hybrid (OFS-1.2) and the next generation OFS switch (OFS-1.3) indicate an effort to capture this growing body of wisdom.

##### 5.2 - Distributed controller logic

ForCES was built to distribute the controller logic to autonomous network elements that operate either as ForCES controlled or as integrated hybrid controller.

OFS has created distributed logic per switch, but considers grouping of these switches outside the OFS specifications. The Hybrid [OFS-1.2] provides use cases for Ships-in-the-Night and integrated. The Ships-in-the-Night provide per port allocation to either OFS or standard processing. The Integrated seeks to run both on a set of ports.

### 5.3 - Hybrid controllers

ForCES was built for the hybrid environment where routing and switching protocol.

OFS is now entering the processing of standardizing for hybrid controllers [OFS-1.2].

## 6. Security Considerations

No security considerations.

This is an informational comparison used to inform clarify ForCES work.

## 7. IANA Considerations

No IANA considerations.

## 8. Conclusions

Both ForCES and OpenFlow follow the basic idea of separations of forwarding plane and control plane in network elements. Both are capable of operating for centralized control, distributed control, and hybrid control.

[OFS-1.1] Flow Table Logic with the instructions and Group Tables is the major difference between the ForCES RFCs. As this paper has shown, the full ramifications of this difference need to be considered in terms of differences in capability of implementation. The author welcomes any additional implementation experience.

[OFS-1.0][OFS-1.1] lacks a forwarding model, a standardized LFB library and the concepts of FE-CE associations (FE-Manger, CE-Manager, pre/post association phase). It appears the OpenFlow work is starting to invent the equivalent of existing ForCES work as OpenFlow work. The guide of this reinventing seems to be the Google code snippets passed to the OpenFlow Forum as examples of "running code" to provide rough consensus.

## 9. References

### 9.1. Normative References

- [RFC2119] Bradner, S., "Key words for use in RFCs to Indicate Requirement Levels", BCP 14, RFC 2119, March 1997.
- [RFC5810] Doria, A., Hadi Salim, J., Haas, R., Khosravi, H., Wang, W., Dong, L., Gopal, R., and J. Halpern, "Forwarding and Control Element Separation (ForCES) Protocol Specification", RFC 5810, March 2010.
- [RFC5812] Halpern, J. and J. Hadi Salim, "Forwarding and Control Element Separation (ForCES) Forwarding Element Model", RFC 5812, March 2010.
- [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

### 9.2. Informative References

- [McKeown2008]  
McKeown, N., Anderson, T., Balakrishnan, H., et al,  
"Openflow: enabling innovation in campus networks", ACM  
SIGCOMM Computer Communication Review. 2008, 38(2):69-  
74.
- [OFS-1.0.0]  
  
OpenFlow Switch Specification - Version 1.0.0 (Wire  
Protocol 0x01), December 31, 2009.
- [OFS-1.0.1-rc3] OpenFlow Switch Errata - Version 1.0.1-r3, June  
12, 2012.

[OFS-1.1.0]

OpenFlow Switch Specification Version 1.1.0 (Wire Protocol 0x02). February 2011.  
(<http://www.openflow.org/documents/openflow-spec-v1.1.0.pdf>)

[OpenFlow-1.2] Open Flow 1.2 - Hybrid Switch (Terminology, Use cases, etc), April-May notes, work-in-progress.

[OFS-1.3-rc4] OpenFlow Switch Specification 1.3 (version 1.3-rc4) (Wire Protocol 0x04) April 4, 2012. [OpenFlow members only]

[OFS-1.1.0]

OpenFlow Switch Specification Version 1.1.0 (Wire Protocol 0x02). February 2011.  
(<http://www.openflow.org/documents/openflow-spec-v1.1.0.pdf>)

[OFC-1.0] OpenFlow Configuration and Management Protocol OF-CONFIG 1.0 (January 15, 2012).

[RFC3654] Khosravi, H. and T. Anderson, "Requirements for Separation of IP Control and Forwarding", RFC 3654, November 2003.

[RFC3746] Yang, L., Dantu, R., Anderson, T., and R. Gopal, "Forwarding and Control Element Separation (ForCES) Framework", RFC 3746, April 2004.

[LFB-Lib] Wang W., Haleplidis E., Ogawa K., Li C., J. Halpern, "ForCES Logical Function Block (LFB) Library", draft-ietf-forces-lfb-lib-06, Work in Progress.

## 10. Acknowledgments

The author would like to thank Tina Tsou, Zhiliang Wang, Jing Huang, Xingang Shi, and Xia Yin. Their earlier draft comparing the FORCES and ONF Technology inspired this draft providing an opposing viewpoint. It is said that "iron sharpens iron" so that in our debates we sharpen our understandings.

The author also acknowledges Edward Crabbe's succinct comments which also inspired the in-depth comparison found in this draft. I only hope that this "wordy" draft proves worth of his pithy and concise insights.

Authors' Addresses

Susan Hares  
Huawei Technologies (USA)  
2330 Central Expressway  
Santa Clara, CA 95050  
USA

Email: Susan Hares [Susan.Hares@huawei.com](mailto:Susan.Hares@huawei.com)