|  |
| --- |
|  |
| OF-CONFIG 1.1 |
| OpenFlow Management and Configuration Protocol |
| Document Version .09 |
|  |
|  |

Contact: [email addresses of people to contact]

Copyright © 2012 Open Networking Foundation

Disclaimer

THIS SPECIFICATION IS PROVIDED "AS IS" WITH NO WARRANTIES WHATSOEVER, INCLUDING ANY WARRANTY OF MERCHANTABILITY, NONINFRINGEMENT, FITNESS FOR ANY PARTICULAR PURPOSE, OR ANY WARRANTY OTHERWISE ARISING OUT OF ANY PROPOSAL, SPECIFICATION OR SAMPLE. Without limitation, ONF disclaims all liability, including liability for infringement of any proprietary rights, relating to use of information in this specification and to the implementation of this specification, and ONF disclaims all liability for cost of procurement of substitute goods or services, lost profits, loss of use, loss of data or any incidental, consequential, direct, indirect, or special damages, whether under contract, tort, warranty or otherwise, arising in any way out of use or reliance upon this specification or any information herein.

No license, express or implied, by estoppel or otherwise, to any Open Networking Foundation or Open Networking Foundation member intellectual property rights is granted herein.

Except that a license is hereby granted by ONF to copy and reproduce this specification for internal use only.

Contact the Open Networking Foundation at <http://www.opennetworking.org>for information on specification licensing through membership agreements.

Any marks and brands contained herein are the property of their respective owners.

Contents

[1 Introduction 6](#_Toc318908474)

[2 Motivation 6](#_Toc318908475)

[3 Scope 8](#_Toc318908476)

[4 Normative Language 8](#_Toc318908477)

[5 Terms 8](#_Toc318908478)

[5.1 OpenFlow Capable Switch 8](#_Toc318908479)

[5.2 OpenFlow Configuration Point 9](#_Toc318908480)

[5.3 OpenFlow Logical Switch 9](#_Toc318908481)

[5.4 OpenFlow Resource 9](#_Toc318908482)

[5.4.1 OpenFlow Queue 9](#_Toc318908483)

[5.4.2 OpenFlow Port 9](#_Toc318908484)

[5.5 OpenFlow Controller 9](#_Toc318908485)

[6 Requirements 9](#_Toc318908486)

[6.1 Requirements from the OpenFlow 1.2 Protocol Specification 9](#_Toc318908487)

[6.1.1 Connection Setup to a Controller 10](#_Toc318908488)

[6.1.2 Multiple Controllers 10](#_Toc318908489)

[6.1.3 OpenFlow Logical Switches 10](#_Toc318908490)

[6.1.4 Connection Interruption 10](#_Toc318908491)

[6.1.5 Encryption 10](#_Toc318908492)

[6.1.6 Queues 11](#_Toc318908493)

[6.1.7 Ports 11](#_Toc318908494)

[6.1.8 Datapath ID 12](#_Toc318908495)

[6.2 Operational Requirements 12](#_Toc318908496)

[6.3 Requirements for the Switch Management Protocol 12](#_Toc318908497)

[7 Data Model 13](#_Toc318908498)

[7.1 OpenFlow Capable Switch 15](#_Toc318908499)

[7.1.1 UML Diagram 15](#_Toc318908500)

[7.1.2 XML Schema 15](#_Toc318908501)

[7.1.3 XML Example 16](#_Toc318908502)

[7.1.4 Normative Constraints 17](#_Toc318908503)

[7.1.5 YANG Specification 17](#_Toc318908504)

[7.2 OpenFlow Configuration Point 18](#_Toc318908505)

[7.2.1 UML Diagram 18](#_Toc318908506)

[7.2.2 XML Schema 18](#_Toc318908507)

[7.2.3 XML Example 19](#_Toc318908508)

[7.2.4 Normative Constraints 19](#_Toc318908509)

[7.2.5 YANG Specification 19](#_Toc318908510)

[7.3 OpenFlow Logical Switch 20](#_Toc318908511)

[7.3.1 UML Diagram 21](#_Toc318908512)

[7.3.2 XML Schema 21](#_Toc318908513)

[7.3.3 XML Example 22](#_Toc318908514)

[7.3.4 Normative Constraints 22](#_Toc318908515)

[7.3.5 YANG Specification 23](#_Toc318908516)

[7.4 Logical Switch Capabilities 24](#_Toc318908517)

[7.4.1 UML Diagram 24](#_Toc318908518)

[7.4.2 XML Schema 24](#_Toc318908519)

[7.4.3 XML Example 30](#_Toc318908520)

[7.4.4 Normative Text 31](#_Toc318908521)

[7.4.5 Yang Specification 31](#_Toc318908522)

[7.5 OpenFlow Controller 31](#_Toc318908523)

[7.5.1 UML Diagram 32](#_Toc318908524)

[7.5.2 XML Schema 33](#_Toc318908525)

[7.5.3 XML Example 34](#_Toc318908526)

[7.5.4 Normative Constraints 34](#_Toc318908527)

[7.5.5 YANG Specification 35](#_Toc318908528)

[7.6 OpenFlow Resource 37](#_Toc318908529)

[7.6.1 UML Diagram 37](#_Toc318908530)

[7.6.2 XML Schema 37](#_Toc318908531)

[7.6.3 XML Example 37](#_Toc318908532)

[7.6.4 Normative Constraints 37](#_Toc318908533)

[7.6.5 YANG Specification 37](#_Toc318908534)

[7.7 OpenFlow Port 38](#_Toc318908535)

[7.7.1 UML Diagram 38](#_Toc318908536)

[7.7.2 XML Schema 39](#_Toc318908537)

[7.7.3 XML Example 40](#_Toc318908538)

[7.7.4 Normative Constraints 41](#_Toc318908539)

[7.7.5 YANG Specification 42](#_Toc318908540)

[7.8 OpenFlow Port Feature 44](#_Toc318908541)

[7.8.1 UML Diagram 45](#_Toc318908542)

[7.8.2 XML Schema 45](#_Toc318908543)

[7.8.3 XML Example 46](#_Toc318908544)

[7.8.4 Normative Constraints 46](#_Toc318908545)

[7.8.5 YANG Specification 47](#_Toc318908546)

[7.9 OpenFlow Queue 48](#_Toc318908547)

[7.9.1 UML Diagram 49](#_Toc318908548)

[7.9.2 XML Schema 49](#_Toc318908549)

[7.9.3 XML Example 50](#_Toc318908550)

[7.9.4 Normative Constraints 50](#_Toc318908551)

[7.9.5 YANG Specification 51](#_Toc318908552)

[7.10 External Certificate 52](#_Toc318908553)

[7.10.1 UML Diagram 52](#_Toc318908554)

[7.10.2 XML Schema 52](#_Toc318908555)

[7.10.3 XML Example 52](#_Toc318908556)

[7.10.4 Normative Constraints 52](#_Toc318908557)

[7.10.5 YANG Specification 53](#_Toc318908558)

[7.11 Owned Certificate 53](#_Toc318908559)

[7.11.1 UML Diagram 53](#_Toc318908560)

[7.11.2 XML Schema 53](#_Toc318908561)

[7.11.3 XML Example 53](#_Toc318908562)

[7.11.4 Normative Constraints 53](#_Toc318908563)

[7.11.5 YANG Specification 54](#_Toc318908564)

[8 Binding to NETCONF 54](#_Toc318908565)

[8.1 How Data Model is Bound to Netconf 56](#_Toc318908566)

[8.1.1 edit-config 56](#_Toc318908567)

[8.1.2 get-config 58](#_Toc318908568)

[8.1.3 copy-config 59](#_Toc318908569)

[8.1.4 delete-config 59](#_Toc318908570)

[8.2 RPC error 60](#_Toc318908571)

[Appendix A XMLSchema 61](#_Toc318908572)

[Appendix B YANG Specification 71](#_Toc318908573)

[Appendix C Bibliography 80](#_Toc318908574)

[Appendix D Revision History 81](#_Toc318908575)

[Appendix E Considerations for Next or Future Releases 81](#_Toc318908576)

[Appendix F Appendix E: ToDo List 81](#_Toc318908577)

[Appendix G Notes From The Board And The Tag 84](#_Toc318908578)

# Introduction

This document describes the motivation, scope, requirements, and specification of the standard configuration and management protocol of an operational context which is capable of containing an OpenFlow 1.2 switch as described in Figure 1. This configuration and management protocol is referred to as OF-CONFIG and is a companion protocol to OpenFlow. This document specifies version 1.0 of OF-CONFIG.

**OpenFlow**

**Switch**

**OpenFlow**

**Controller**

**OpenFlow**

**Protocol**

**Operational Context**

**OF**

**-**

**CONFIG**

**OpenFlow   
Configuration  
Point**

pdates based on 12/22/callto co

Figure : An OpenFlow Configuration Point communicates with an operational context which is capable of supporting an OpenFlow Switch using the OpenFlow Configuration and Management Protocol (OF-CONFIG)

The reader of this document is assumed to be familiar with the OpenFlow protocol and OpenFlow related concepts. Reading the OpenFlow whitepaper (2) and the OpenFlow Specification (1) is recommended prior to reading this document.

It is strongly recommended that switches which implement OF-CONFIG make changes to the OpenFlow operational context described in this document via OF-CONFIG and limit changes to the OpenFlow operational context via other methods (e.g. command line interfaces and other legacy management protocols). Future versions may better support other methods of change with detailed notification to the OpenFlow Configuration Point via OF-CONFIG.

# Motivation

The OpenFlow protocol assumes that an OpenFlow datapath (e.g. an Ethernet switch which supports the OpenFlow protocol) has been configured with various artifacts such as the IP addresses of OpenFlow controllers. The motivation for the OpenFlow Configuration Protocol (OF-CONFIG) is to enable the remote configuration of OpenFlow datapaths. While the OpenFlow protocol generally operates on a time-scale of a flow (i.e. as flows are added and deleted), OF-CONFIG operates on a slower time-scale.

OF-CONFIG frames an OpenFlow datapath as an abstraction called an OpenFlow Logical Switch. The OF-CONFIG protocol enables configuration of essential artifacts of an OpenFlow Logical Switch so that an OpenFlow controller can communicate and control the OpenFlow Logical switch via the OpenFlow protocol.

OF-CONFIG 1.0 introduces an operating context for one or more OpenFlow datapaths called an OpenFlow Capable Switch. An OpenFlow Capable Switch is intended to be equivalent to an actual physical or virtual network element (e.g. an Ethernet switch) which is hosting one or more OpenFlow datapaths by partitioning a set of OpenFlow related resources such as ports and queues among the hosted OpenFlow datapaths. The OF-CONFIG protocol enables dynamic association of the OpenFlow related resources of an OpenFlow Capable Switch with specific OpenFlow Logical Switches which are being hosted on the OpenFlow Capable Switch. OF-CONFIG does not specify or report how the partitioning of resources on an OpenFlow Capable Switch is achieved. OF-CONFIG assumes that resources such as ports and queues are partitioned amongst multiple OpenFlow Logical Switches such that each OpenFlow Logical Switch can assume full control over the resources that is assigned to it.

OF-CONFIG 1.0 makes simplifying assumptions about the architecture of OpenFlow switches. The specification is deliberately decoupled from whether the switch supports flowvisor or other virtualization models.

The service which sends OF-CONFIG messages to an OpenFlow Capable Switch is called an OpenFlow Configuration Point. No assumptions are made about the nature of the OpenFlow Configuration Point. For example, it may be a service provided by software acting as an OpenFlow controller or it may by a service provided by a traditional network management framework. Any interaction between the OpenFlow Configuration Points and OpenFlow controllers is outside the scope of OF-CONFIG 1.0.

Figure 2 shows the basic abstractions detailed in OF-CONFIG 1.0 and the lines indicate that the OpenFlow Configuration Points and OpenFlow Capable Switches communicate via OF-OFCONFIG while OpenFlow Controllers and OpenFlow Logical Switches (i.e. datapaths) communicate via OpenFlow.



Figure : Relationship between components defined in this specification, the OF-CONFIG protocol and the OpenFlow protocol

A guiding principle in the development of this specification is to keep the protocol and schema simple and leverage existing protocols and schema models where possible. This helped in quick development of this specification and hopefully will also enable easier adoption, the motivation being to supplement the OpenFlow specification in a meaningful way to further drive the adoption of the software defined networking vision.

# Scope

OF-CONFIG 1.0 is focused on the basic functions needed to configure an OpenFlow 1.2 (OFv1.2) datapath. Functionality to be configured includes:

* The assignment of one or more OpenFlow controllers
* The configuration of queues and ports
* The ability to remotely change some aspects of ports (e.g. up/down)

While limited in scope, OF-CONFIG1.0 lays the foundation on top of which various automated and more advanced configurations will be possible in future revisions. Tunnel configuration, switch discovery, topology discovery, capability reporting, event triggers, instantiation of OpenFlow Logical Switches, assignment of resources such as ports and queues to OpenFlow Logical Switches, and bootstrap of the OpenFlow capable network are outside the scope of OF-CONFIG 1.0 protocol. These may be included in future versions.

# Normative 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 RFC 2119 (3).

# Terms

The following section lists several terms and definitions used in this document.

## OpenFlow Capable Switch

An OpenFlow Capable switch is a physical or virtual switching device which can act an as operational context for an OpenFlow Logical Switch. OpenFlow Capable Switches contain and manage OpenFlow Resources which may be associated with an OpenFlow Logical Switch context.

## OpenFlow Configuration Point

An OpenFlow Configuration Point configures one or more OpenFlow Capable Switches via the OpenFlow Configuration and Management Protocol (OF-CONFIG).

## OpenFlow Logical Switch

An OpenFlow Logical Switch is a set of resources (e.g. ports) from an OpenFlow Capable Switch which can be associated with a specific OpenFlow Controller. An OpenFlow Logical switch is an instantiation of an OpenFlow Datapath as specified in (1).

## OpenFlow Resource

An OpenFlow Resource is a resource (e.g. port or queue) which is associated with an OpenFlow Capable Switch and may be associated with an OpenFlow Logical Switch.

### OpenFlow Queue

An OpenFlow Queue is a queuing resource of an OpenFlow Logical Switch as described in the OpenFlow specification as the queue component of an OpenFlow datapath.

### OpenFlow Port

An OpenFlow Port is a forwarding interface of an OpenFlow Logical Switch as described in the OpenFlow specification as the port component of an OpenFlow datapath.

## OpenFlow Controller

An OpenFlow Controller is software which controls OpenFlow Logical Switches via the OpenFlow protocol.

# Requirements

This section describes requirements for the design of OF-CONFIG 1.0.

## Requirements from the OpenFlow 1.2 Protocol Specification

The specification of version 1.2 of the OpenFlow protocol (1) includes explicit and implicit requirements for the configuration of OpenFlow switches. In (1) the term ‘configuration’ is used for two different kinds of operations: Configuration using the OpenFlow protocol and configuration outside of the OpenFlow protocol. The first kind of configuration is dealt with in (1). OF-CONFIG 1.0 enables other configuration of OpenFlow switches.

### Connection Setup to a Controller

Section 6.2 (Connection Setup) of (1) discusses the process of setting up a connection between the OpenFlow switch and an OpenFlow controller. The switch initiates the connection applying three parameters that need to be configured in advance:

* the IP address of the controller
* the port number at the controller
* the transport protocol to use, either TLS or TCP

OF-CONFIG 1.0 must provide means for configuring these parameters.

### Multiple Controllers

Section 6.3 of (1) discusses how a switch deals with multiple controllers simultaneously. This implicitly requires OF-CONFIG 1.0 to provide means for configuring multiple instances of the parameter set listed in 6.1.1 for specifying the connection setup to multiple controllers.

### OpenFlow Logical Switches

The OpenFlow 1.2 protocol specifies various kinds of OpenFlow resources associated with an OpenFlow Logical Switch. The OF-CONFIG protocol must support the configuration of these OpenFlow resources associated with an OpenFlow Logical Switch. Examples of resources include queues and ports that have been assigned to an OpenFlow Logical Switch. It is assumed that OpenFlow Logical Switches have been instantiated out of band, for example, an administrator may have created them upfront. In addition, partitioning/assignment of OpenFlow resources amongst multiple OpenFlow switches that may exist in an OpenFlow Capable Switch has also been done out of band.

### Connection Interruption

Section 6.4 of (1) discusses the choice of two modes the switch should immediately enter after losing contact with all controllers. The modes are

* fail secure mode
* fail standalone mode

OF-CONFIG protocol must provide means for configuring the mode to enter in such a case.

### Encryption

Section 6.4 of (1) discusses encryption of connections to controllers that use TLS. It explicitly states “Each switch must be user-configurable with one certificate for authenticating the controller (controller certificate) and the other for authenticating to the controller (switch certificate)”. Hence, OF-CONFIG must provide means for configuring a switch certificate and a controller certificate for each controller that is configured to use TLS. This requirement is not addressed in OF-CONFIG 1.0 and will be addressed in a future version.

### Queues

Section A.3.6 of (1) the configuration of queues. Queue in (1) have three parameters that may be configurable:

* min-rate
* max-rate
* experimenter

OF-CONFIG 1.0 must provide means for configuring these parameters.

### Ports

The OpenFlow protocol already contains methods to configure ports of OpenFlow switches. The OpenFlow protocol specification (1) does not explicitly require an external configuration means, and therefore we cannot derive the requirement for configuring ports from (1). However, the configuration of ports is an essential step of configuring a network and thus an obvious requirement for OF-CONFIG 1.0. Section A.2.1 of (1) defines the following parameters for port configuration:

* current-speed
* no-recv
* no-fwd
* no-packet-in
* link-down
* blocked
* live

OF-CONFIG 1.0 must provide means for configuring these parameters.

Also defined in the OpenFlow protocol specification are port features. There are four sets of these features for current, advertised, supported, and peer-advertised features. Feature sets current, supported, and peer-advertised contain state information and are not to be configured. Only advertised features could potentially be configured with the following parameters:

* speed
* duplex-mode
* copper-medium
* fiber-medium
* auto-negotiation
* pause
* asymmetric-pause

OF-CONFIG 1.0 must provide means for configuring these advertised features.

### Datapath ID

Section A.3.1 of (1) discusses the datapath ID of a switch. It is a 64-bit filed with the lower 48 bit intended for the switch MAC address and the remaining 16 bit left to the switch operator. Although not explicitly requested by (1), OF-CONFIG should provide means for configuring the datapath ID.

## Operational Requirements

The OF-CONFIG 1.0 must meet support the following scenarios:

1. OF-CONFIG 1.0 must support an OpenFlow Capable Switch being configured by multiple OpenFlow Configuration Points.
2. OF-CONFIG 1.0 must support an OpenFlow Configuration Point managing multiple OpenFlow Capable Switches.
3. OF-CONFIG 1.0 must support an OpenFlow Logical Switch being controlled by multiple OpenFlow Controllers.
4. OF-CONFIG 1.0 must support configuring ports and queues of an OpenFlow Capable Switch that have been assigned to an OpenFlow Logical Switch.

## Requirements for the Switch Management Protocol

OF-CONFIG 1.0 defines a communication standard between an OpenFlow switch and an OpenFlow Configuration Point. It consists of a network management protocol specified in Section 8 and a data model defined in Section 7. This subsection specifies requirements for the network management protocol. The protocol must comply with the following requirements:

1. The protocol must be secure providing integrity, privacy, and authentication. Authentication of both ends, switch and configuration point, must be supported.
2. The protocol must support reliable transport of configuration requests and replies.
3. The protocol must support connection setup by the configuration point.
4. The protocol should support connection setup by the switch.
5. The protocol must be able to carry partial switch configurations.
6. The protocol must be able to carry bulk switch configurations.
7. The protocol must support the configuration point setting configuration data at the switch
8. The protocol must support the configuration point retrieving configuration data from the switch.
9. The protocol should support the configuration point retrieving status information from the switch.
10. The protocol must support creation, modification and deletion of configuration information at the switch.
11. The protocol must support reporting on the result of a successful configuration request.
12. The protocol must support reporting error codes for partially or completely failed configuration requests.
13. The protocol should support sending configuration requests independent of the completion of previous requests.
14. The protocol should support transaction capabilities including rollback per operation.
15. The protocol must provide means for asynchronous notifications from the switch to the configuration point.
16. The protocol should be extensible.
17. The protocol should support reporting its capabilities.

# Data Model

This section specifies the data model for OF-CONFIG 1.0. Configurations of an OpenFlow Capable Switch or for portions of it are encoded in XML. The data model is structured into classes and attributes of classes. Each class is described in a separate sub-section by

1. a UML diagram giving an overview of the class,
2. a portion of an XML schemaextracted from the normative XML schema in Appendix A,
3. an example for XML code encoding an instance of the class,
4. normative constraints for instances of the classextending the XML schema by semantic specifications,
5. a portion of a YANG (9) moduleextracted from the YANG module in Appendix B.

The full XML schema and the full YANG module are listed in Appendices A and B. Normative for OF-CONFIG 1.0 is the XML schema in Appendix A and the normative constraints in sub-sections 7.X.4. The YANG module in Appendix B incorporates the XML schema specifications as well as the normative constraints.

OF-CONFIG specific terminology used for describing the model is defined in Section 5. The following UML diagram describes the top-level classes of the data model.



Figure : UML Class Diagram for OF-CONFIG Data Model

The core of the model is an OpenFlow Capable Switch that is configured by OpenFlow Configuration Points.

The switch contains a set of resources of different types. For OF-CONFIG 1.0, two types of resources are included in the model: OpenFlow Ports and OpenFlow Queues. More resource types may be added in future revisions of OF-CONFIG. OpenFlow resources can be made available for use to OpenFlow Logical Switches.

Instances of OpenFlow logical switches are contained within the OpenFlow Capable Switch. A set of OpenFlow Controllers is assigned to each OpenFlow logical switch.

The data model contains several identifiers, most of them encoded as an XML element <id>. Currently these IDs are defined as strings with required uniqueness in a certain context. Beyond uniqueness requirements, no further guidance is given on how to build these strings. This may be changed in the future. Particularly, the use of Universal Resource Names (URNs) is envisioned. This requires developing a naming scheme for URNs in OF-CONFIG and registering a URN namespace for the ONF. It is expected that recommendations for URN-based identifiers will be introduced by a future version of OF-CONFIG. Since URNs are represented as strings, such recommendations can be made compatible with identifiers in OF-CONFIG v1.0.

When issuing a NETCONF get request all elements in the requested sub-tree must be returned in the result. Those elements that can be modified by a NETCONF edit-config request or retrieved by a NETCONF get-config request are identified in the normative constraints sub-sections 8.X.4.

## OpenFlow Capable Switch

The OpenFlow Capable Switch serves as the root element for an OpenFlow configuration. It has relationships to

* OpenFlow Configuration Points that manage and particularly configure the OpenFlow Capable Switch,
* OpenFlow logical switches that are contained and instantiated within the OpenFlow Capable Switch,
* OpenFlow Resources contained in the OpenFlow Capable Switch that may be used by OpenFlow Logical Switches.

### UML Diagram



Figure : Data Model Diagram for OpenFlow Capable Switch

### XML Schema

|  |
| --- |
| <xs:complexType name="OFCapableSwitchType">  <xs:sequence>  <xs:element name="id"  type="OFConfigID"/>  <xs:element name="configuration-points"  type="OFConfigurationPointListType"/>  <xs:element name="resources"  type="OFCapableSwitchResourceListType"/>  <xs:element name="logical-switches"  type="OFLogicalSwitchListType"/>  </xs:sequence>  </xs:complexType>  <xs:complexType name="OFConfigurationPointListType">  <xs:sequence>  <xs:element name=“configuration-point”  type="OFConfigurationPointType"  maxOccurs="unbounded"/>  </xs:sequence>  </xs:complexType>  <xs:complexType name="OFCapableSwitchResourceListType">  <xs:sequence>  <xs:element name="port"  type="OFPortType" maxOccurs="unbounded"/>  <xs:element name="queue"  type="OFQueueType" maxOccurs="unbounded"/>  <xs:element name="owned-certificate"  type="OFOwnedCertificateType" maxOccurs="unbounded"/>  <xs:element name="external-certificate"  type="OFExternalCertificateType"  maxOccurs="unbounded"/>  <xs:element name="flow-table"  type="OFFlowTableType" maxOccurs="unbounded"/>  </xs:sequence>  </xs:complexType>  <xs:complexType name="OFLogicalSwitchListType">  <xs:sequence>  <xs:element name="logical-switch"  type="OFLogicalSwitchType"  maxOccurs="unbounded"/>  </xs:sequence>  </xs:complexType> |

### XML Example

|  |
| --- |
| <capable-switch>  <id>CapableSwitch0</id>  <configuration-points>  ...  </configuration-points>  <resources>  ...  </resources>  <logical-switches>  ...  </logical-switches>  </capable-switch> |

### Normative Constraints

The OpenFlow Capable Switch is identified by the OpenFlow Configuration Point with identifier <id>. The identifier MUST be unique within the context of potential OpenFlow Configuration Points. It MUST be persistent across reboots of the OpenFlow Capable Switch.

Element <configuration-points>contains a list of all Configuration Points known to the OpenFlow Capable Switch that manage it or have managed it using OF-CONFIG.

Element <resources> contains lists of all resources of an OpenFlow Capable Switch that can be used by OpenFlow Logical Switches. Resources are listed here independent of their actual assignment to OpenFlow Logical Switches. They may be available to be assigned to an OpenFlow Logical Switch or already in use by an OpenFlow Logical Switch.

Element <logical-switches> contains a list of all OpenFlow Logical Switches available on the OpenFlow Capable Switch.

### YANG Specification

|  |
| --- |
| container capable-switch {  description "The OpenFlow Capable Switch containing logical switches, and resources that can be assigned to logical switches.";  leaf id {  type inet:uri;  mandatory true;  description "An unique but locally arbitrary identifier that identifies a Capable Switch towards the management system and is persistent across reboots of the system.";  }  container configuration-points {  list configuration-point {  key "id";  unique "id";  description "The list of all Configuration Points known to the OpenFlow Capable Switch that may configure it using OF-CONFIG.";  uses openflow-configuration-point-grouping;  }  }  container resources {  description "A lists containing all resources of the OpenFlow Capable Switch.";  ...  }  container logical-switches {  description "This element contains all OpenFlow Logical Switches on the OpenFlow Capable Switch.";  list switch {  key "id";  unique "id";  description "The list of all OpenFlow Logical Switches the OpenFlow Capable Switch.";  uses openflow-logical-switch-grouping;  }  }  } |

## OpenFlow Configuration Point

The Configuration Point is an entity that manages the switch using the OF-CONFIG protocol. Attributes of an OpenFlow Configuration Point allow the OpenFlow Capable Switches to identify a Configuration Point and specify which protocol is used for communication between Configuration Point and OpenFlow Capable Switch. The OpenFlow Capable Switch stores a list of Configuration Points that manage it or have managed it. An OpenFlow Configuration Point is to an OpenFlow Capable Switch what an OpenFlow Controller is to an OpenFlow Logical switch.

Instances of the Configuration Point class are used by switches to connect to a configuration point. Currently the only transport mapping that supports a connection set-up initiated by the switch to be configured is the mapping to the BEEP protocol (5). Other NETCONF transport mappings (6,7,8) may be extended in the future to also support connection set-up in this direction.

### UML Diagram



Figure : Data Model Diagram for an OpenFlow Configuration Point

### XML Schema

|  |
| --- |
| <xs:complexType name="OFConfigurationPointType">  <xs:sequence>  <xs:element name="id"  type="OFConfigID"/>  <xs:element name="uri"  type="inet:uri"/>  <xs:element name="protocol"  type="OFConfigurationPointProtocolType"/>  </xs:sequence>  </xs:complexType>  <xs:simpleType name="OFConfigurationPointProtocolType">  <xs:restriction base="xs:string">  <xs:enumeration value="ssh"/>  <xs:enumeration value="soap"/>  <xs:enumeration value="tls"/>  <xs:enumeration value="beep"/>  </xs:restriction>  </xs:simpleType> |

### XML Example

|  |
| --- |
| <configuration-point>  <id>ConfigurationPoint1</id>  <uri>uri0</uri>  <protocol>ssh</protocol>  <configuration-point> |

### Normative Constraints

OF-CONFIG uses the NETCONF protocol as described in Section 8. NETCONF can use four different transport protocols: SSH, BEEP, SOAP, and TLS. Element <protocol> defines the transport protocol that the Configuration Point used last when communicating via NETCONF with the OpenFlow Capable Switch. If this element is missing, then the default protocol is SSH.

When an OpenFlow Capable Switch connects to a configuration point it must make sure that the used connection information is stored in an instance of the Configuration Point class. If such an instance does not exist, the OpenFlow Capable Switch MUST create an instance that it fills with connection information.

An OpenFlow Capable Switch that cannot initiate a connection to a configuration point does not have to implement the Configuration Point class. It SHOULD block attempts to write to instances of the Configuration Point class with NETCONF <edit-config>operations.

Instances of the Configuration Point class SHOULD be stored persistently across reboots of the OpenFlow Capable Switch.

A Configuration Point is identified by OpenFlow Capable Switches with identifier <id>. The identifier MUST be unique within the context of potential OpenFlow Capable Switches.

Element <uri> identifies the location of the configuration point as a service resource and MUST include all information necessary for the OpenFlow Capable Switch to reconnect to the Configuration Point should it become disconnected (e.g. protocol, fully qualified domain name, and port).

The following elements of the Configuration Point can be modified by a NETCONF edit-config request or retrieved by a NETCONF get-config request: <id>, <uri>, <protocol>.

### YANG Specification

|  |
| --- |
| grouping openflow-configuration-point-grouping {  description "Representation of an OpenFlow Configuration Point.";  leaf id {  type inet:uri;  description "An identifier that identifies a Configuration Point of the OpenFlow Capable Switch.";  }  leaf uri {  type inet:uri;  description "A locator of the Configuration Point. This element MAY contain a locator of the Configuration Point including, for example, an IP address and a port number.";  }  leaf protocol {  type enumeration {  enum "ssh";  enum "soap";  enum "tls";  enum "beep";  }  default "ssh";  description "The transport protocol that the Configuration Point uses when communicating via NETCONF with the OpenFlow Capable Switch.";  reference "The mappings of NETCONF to different transport protocols are defined in RFC 6242 for SSH, RFC 4743 for SOAP, RFC 4744 for BEEP, and RFC 5539 for TLS";  }  } |

## OpenFlow Logical Switch

The OpenFlow Logical Switch represents an instant of a logical switch that is available or can be made available on an OpenFlow Capable Switch. An OpenFlow Logical switch is a logical context which behaves as the datapath as described in the OpenFlow specification. The OpenFlow Logical Switch is connected to one or more OpenFlow Controllers via the OpenFlow protocol. It uses resources of the OpenFlow Capable Switch for realizing the capabilities offered via the OpenFlow protocol. The OpenFlow Logical Switch has relationships to

* OpenFlow Controllers that control the OpenFlow Capable Switch
* OpenFlow Resources that are available from the OpenFlow Capable Switch

### UML Diagram



Figure : Data Model Diagram for an OpenFlow Logical Switch

### XML Schema

|  |
| --- |
| <xs:complexType name="OFLogicalSwitchType">  <xs:sequence>  <xs:element name="id"  type="OFConfigID"/>  <xs:elementname="capabilities"  type="OFLogicalSwitchCapabilitiesType"/>  <xs:element name="datapath-id"  type="OFConfigID"/>  <xs:element name="enabled"  type="xs:boolean"/>  <xs:element name="check-controller-certificate"  type="xs:boolean"/>  <xs:element name="lost-connection-behavior"  type="OFLogicalSwitchLostConnnectionBehavior"/>  <xs:element name="controllers"  type="OFControllerListType"/>  <xs:element name="resources"  type="OFLogicalSwitchResourceListType"/>  </xs:sequence>  </xs:complexType>  <xs:simpleType name="OFLogicalSwitchLostConnnectionBehavior">  <xs:restriction base="xs:string">  <xs:enumeration value="failSecureMode"/>  <xs:enumeration value="failStandaloneMode"/>  </xs:restriction>  </xs:simpleType>  <xs:complexType name="OFControllerListType">  <xs:sequence>  <xs:element name="controller"  type="OFControllerType"  maxOccurs="unbounded"/>  </xs:sequence>  </xs:complexType>  <xs:complexType name="OFLogicalSwitchResourceListType">  <xs:sequence>  <xs:element name="port"  type="OFConfigID" maxOccurs="unbounded"/>  <xs:element name="queue"  type="OFConfigID"  maxOccurs="unbounded"/>  <xs:element name="certificate"  type="OFConfigID" minOccurs="0" maxOccurs="1"/>  <xs:element name="flow-table"  type="OFConfigID"  maxOccurs="unbounded"/>  </xs:sequence>  </xs:complexType> |

### XML Example

|  |
| --- |
| <logical-switch>  <id>LogicalSwitch5</id>  <capabilities>  ...  <capabilities>  <datapath-id>datapath-id0</datapath-id>  <enabled>true</enabled>  <check-controller-certificate>false</check-controller-certificate>  <lost-connection-behavior>failSecureMode</lost-connection-behavior>  <controllers>  ...  </controllers>  <resources>  <port>port2</port>  <port>port3</port>  <queue>queue0</queue>  <queue>queue1</queue>  <certificate>ownedCertificate4</certificate>  <flow-table>1</flow-table>  <flow-table>2</flow-table>  …  <flow-table>255</flow-table>  </resources>  </logical-switch> |

### Normative Constraints

An OpenFlow Logical Switch is identified by identifier <id>. The identifier MUST be unique within the context of the OpenFlow Capable Switch. It MUST be persistent across reboots of the OpenFlow Capable Switch.

Normative constraints for <capabilities> TBD

Element <datapath-id> identifies the OpenFlow Logical Switch to the OpenFlow controllers that has been assigned to the OpenFlow Logical Switch. The <datapath-id> MUST be unique within the context of OpenFlow Controllers associated with OpenFlow Logical Switch. The <datapath-id>is a string value that MUST be formatted as a sequence of 10 2-digit hexadecimal numbers that are separated by colons, e.g.,01:23:45:67:89:ab:cd:ef:01:23. The case of the hexadecimal digits MUST be ignored.

Element <enabled> denotes the administrative state of the OpenFlow Logical Switch. A value of false means the OpenFlow Logical Switch MUST NOT communicate with any OpenFlow Controllers, MUST NOT conduct any OpenFlow processing, and SHOULD NOT be utilizing computational or network resources of the underlying platform.

Element <check-controller-certificate> defines the behavior of the OpenFlow Logical Switch when establishing a connection to a controller. If set to value false, the logical switch will connect to a controller without checking any controller certificate. If set to value true, then the logical switch will connect to a controller with element <protocol> set to TLS, only if the controller provides a certificate that can be verified with one of the certificates stored in the list of <external-certificates> in the OpenFlow Capable Switch.

Element <lost-connection-behavior> defines the behavior of the OpenFlow Logical Switch in case it loses contact with all controllers. Section 6.4 of the OpenFlow specification 1.2 defines two alternative modes in such a case: fails secure mode and fail standalone mode. These are the only allowed values for this element. Default is the fail secure mode.

Element <resources> contains the list of all resources of the OpenFlow Capable Switch that the OpenFlow Logical Switch has exclusive access to. Any resource identified in the <resources>list of a Logical Switch MUST be present in the <resources> list of the OpenFlow Capable Switch containing the OpenFlow Logical Switch. Resources are identified by a <port> element, a <queue> element, a <certificate> element, or a <flow-table> element. Values of these elements MUST match a value of an <resource-id> element of a resource of the OpenFlow Capable Switch.

Any <port>, <queue> or <flow-table> resource identified in the <resources> list of anOpenFlow Logical Switch MUST NOT be identified in the <resources> list of any other OpenFlow Logical Switch.

If there is a <certificate> element present, the logical switch MUST provide the identified certificate when connecting to a controller that has its element <protocol> set to TLS.

The following elements of the OpenFlow Logical Switch can be modified by a NETCONF edit-config request or retrieved by a NETCONF get-config request: <id>, <datapath-id>, <enabled>. Elements in the <resources> list can also be modified and retrieved by those commands.

### YANG Specification

|  |
| --- |
| typedef datapath-id-type {  type string {  pattern  '[0-9a-fA-F]{2}(:[0-9a-fA-F]{2}){7}';  }  description "The datapath-id type represents an OpenFlow datapath identifier.";  }  grouping openflow-logical-switch-grouping {  description "This grouping specifies all properties of an OpenFlow Logical Switch.";  leaf id {  type inet:uri;  mandatory true;  description "An unique but locally arbitrary identifier that identifies a Logical Switch within a Capable Switch and is persistent across reboots of the system.";  }  container capabilities {  description "This container specifies all capability items of an OpenFlow Logical Switch.";  uses openflow-logical-switch-capabilities-grouping;  }  leaf datapath-id {  type datapath-id-type;  mandatory true;  description "The datapath identifier of the Logical Switch that uniquely identifies this Logical Switch in the controller.";  }  leaf enabled {  type boolean;  mandatory true;  description "Specifies if the Logical Switch is enabled.";  }  container controllers {  description "The list of controllers for this Logical switch.";  list controller {  key "id";  unique "id";  description "The list of controllers that are assigned to the OpenFlow Logical Switch.";  uses openflow-controller-grouping;  }  }  container resources {  description "The following lists reference to all resources of the OpenFlow Capable Switch that the OpenFlow Logical Switch has exclusive access to.";  leaf-list port {  type leafref {  path "/capable-switch/resources/port/resource-id";  }  description "The list references to all port resources of the OpenFlow Capable Switch that the OpenFlow Logical Switch has exclusive access to.";  }  leaf-list queue {  type leafref {  path "/capable-switch/resources/queue/resource-id";  }  description "The list references to all queue resources of the OpenFlow Capable Switch that the OpenFlow Logical Switch has exclusive access to.";  }  leaf-list flow-table {  type leafref {  path "/capable-switch/resources/flow-table/resource-id";  }  description "The list references to all flow table resources of the OpenFlow Capable Switch that the OpenFlow Logical Switch has exclusive access to.";  }  }  } |

## Logical Switch Capabilities

### UML Diagram

### XML Schema

|  |
| --- |
| <xs:complexType name="OFLogicalSwitchCapabilitiesType">  <xs:sequence>  <xs:element name="max-buffered-packets" type="xs:integer">  <xs:annotation>  <xs:documentation>The maximum number of packets the switch can buffer when sending packets to the controller using packet-in messages. See OpenFlow protocol 1.2 section A.3.1  </xs:documentation>  </xs:annotation>  </xs:element>  <xs:element name="max-tables" type="xs:integer">  <xs:annotation>  <xs:documentation> The number of flow tables supported by the switch. See OpenFlow protocol 1.2 section A.3.1  </xs:documentation>  </xs:annotation>  </xs:element>  <xs:element name="max-ports" type="xs:integer">  <xs:annotation>  <xs:documentation> The number of ports supported by the switch. See OpenFlow protocol 1.2 section A.3.1  </xs:documentation>  </xs:annotation>  </xs:element>  <xs:element name="flow-statistics" type="xs:boolean">  <xs:annotation>  <xs:documentation> Whether the switch supports flow statistics. See OpenFlow protocol 1.2 section A.3.1  </xs:documentation>  </xs:annotation>  </xs:element>  <xs:element name="table-statistics" type="xs:boolean">  <xs:annotation>  <xs:documentation> Whether the switch supports table statistics. See OpenFlow protocol 1.2 section A.3.1  </xs:documentation>  </xs:annotation>  </xs:element>  <xs:element name="port-statistics" type="xs:boolean">  <xs:annotation>  <xs:documentation>Whether the switch supports port statistics. See OpenFlow protocol 1.2 section A.3.1  </xs:documentation>  </xs:annotation>  </xs:element>  <xs:element name="group-statistics" type="xs:boolean">  <xs:annotation>  <xs:documentation> Whether the switch supports group statistics. See OpenFlow protocol 1.2 section A.3.1  </xs:documentation>  </xs:annotation>  </xs:element>  <xs:element name="queue-statistics" type="xs:boolean">  <xs:annotation>  <xs:documentation>Whether the switch supports queue statistics. See OpenFlow protocol 1.2 section A.3.1  </xs:documentation>  </xs:annotation>  </xs:element>  <xs:element name="reassemble-ip-fragments" type="xs:boolean">  <xs:annotation>  <xs:documentation>Whether the switch supports reassemble IP fragments. See OpenFlow protocol 1.2 section A.3.1  </xs:documentation>  </xs:annotation>  </xs:element>  <xs:element name="block-looping-ports" type="xs:boolean">  <xs:annotation>  <xs:documentation>"true" indicates that a switch protocol outside of OpenFlow, such as 802.1D Spanning Tree, will detect topology loops and block ports to prevent packet loops. See OpenFlow protocol 1.2 section A.3.1  </xs:documentation>  </xs:annotation>  </xs:element>  <xs:element name="reserved-port-types"  type="OFReservedPortTypes">  <xs:annotation>  <xs:documentation>Specify generic forwarding actions such as sending to the controller, ooding, or forwarding using non-OpenFlow methods, such as "normal" switch processing. SeeOpenFlow protocol 1.2 section 4.5.  </xs:documentation>  </xs:annotation>  </xs:element>  <xs:element name="group-types" type="OFGroupTypes">  <xs:annotation>  <xs:documentation>The group types supported by the switch. SeeOpenFlow protocol 1.2 section 5.4.1.  </xs:documentation>  </xs:annotation>  </xs:element>  <xs:element name="group-capabilities" type="OFGroupCapabilities">  <xs:annotation>  <xs:documentation>The group capabilities supported by the switch. SeeOpenFlow protocol 1.2 section A.3.5.9.  </xs:documentation>  </xs:annotation>  </xs:element>  <xs:element name="action-types" type="OFActionTypes">  <xs:annotation>  <xs:documentation>The action types supported by the switch. See OpenFlow protocol 1.2 section 5.9 and A.2.5.  </xs:documentation>  </xs:annotation>  </xs:element>  <xs:element name="instruction-types" type="OFInstructionTypes">  <xs:annotation>  <xs:documentation>The instruction types supported by the switch. See OpenFlow protocol 1.2 section 5.6.  </xs:documentation>  </xs:annotation>  </xs:element>  </xs:sequence>  </xs:complexType>  <xs:complexType name="OFReservedPortTypes">  <xs:sequence minOccurs="1" maxOccurs="unbounded">  <xs:element name="type" type="OFReservedPortType"/>  </xs:sequence>  </xs:complexType>  <xs:simpleType name="OFReservedPortType">  <xs:restriction base="xs:string">  <xs:enumeration value="all"/>  <xs:enumeration value="controller"/>  <xs:enumeration value="table"/>  <xs:enumeration value="inport"/>  <xs:enumeration value="any"/>  <xs:enumeration value="local"/>  <xs:enumeration value="normal"/>  <xs:enumeration value="flood"/>  </xs:restriction>  </xs:simpleType>  <xs:complexType name="OFGroupTypes">  <xs:sequence minOccurs="1" maxOccurs="unbounded">  <xs:element name="type" type="OFGroupType"/>  </xs:sequence>  </xs:complexType>  <xs:simpleType name="OFGroupType">  <xs:restriction base="xs:string">  <xs:enumeration value="all"/>  <xs:enumeration value="select"/>  <xs:enumeration value="indirect"/>  <xs:enumeration value="fast-failover"/>  </xs:restriction>  </xs:simpleType>  <xs:complexType name="OFGroupCapabilities">  <xs:sequence minOccurs="1" maxOccurs="unbounded">  <xs:element name="capability" type="OFGroupCapability"/>  </xs:sequence>  </xs:complexType>  <xs:simpleType name="OFGroupCapability">  <xs:restriction base="xs:string">  <xs:enumeration value="select-weight"/>  <xs:enumeration value="select-liveness"/>  <xs:enumeration value="chaining"/>  <xs:enumeration value="chaining-check"/>  </xs:restriction>  </xs:simpleType>  <xs:complexType name="OFActionTypes">  <xs:sequence minOccurs="1" maxOccurs="unbounded">  <xs:element name="type" type="OFActionType"/>  </xs:sequence>  </xs:complexType>  <xs:simpleType name="OFActionType">  <xs:restriction base="xs:string">  <xs:enumeration value="output"/>  <xs:enumeration value="copy-ttl-out"/>  <xs:enumeration value="copy-ttl-in"/>  <xs:enumeration value="set-mpls-ttl"/>  <xs:enumeration value="dec-mpls-ttl"/>  <xs:enumeration value="push-vlan"/>  <xs:enumeration value="pop-vlan"/>  <xs:enumeration value="push-mpls"/>  <xs:enumeration value="pop-mpls"/>  <xs:enumeration value="set-queue"/>  <xs:enumeration value="group"/>  <xs:enumeration value="set-nw-ttl"/>  <xs:enumeration value="dec-nw-ttl"/>  <xs:enumeration value="pop-mpls"/>  <xs:enumeration value="set-field"/>  </xs:restriction>  </xs:simpleType>  <xs:complexType name="OFInstructionTypes">  <xs:sequence minOccurs="1" maxOccurs="unbounded">  <xs:element name="type" type="OFInstructionType"/>  </xs:sequence>  </xs:complexType>  <xs:simpleType name="OFInstructionType">  <xs:restriction base="xs:string">  <xs:enumeration value="apply-actions"/>  <xs:enumeration value="clear-actions"/>  <xs:enumeration value="write-actions"/>  <xs:enumeration value="write-metadata"/>  <xs:enumeration value="goto-table"/>  </xs:restriction>  </xs:simpleType> |

### XML Example

|  |
| --- |
| <capabilities>  <max-buffered-packets>512</max-buffered-packets>  <max-tables>1024</max-tables>  <max-ports>2048</max-ports>  <flow-statistics>true</flow-statistics>  <table-statistics>false</table-statistics>  <port-statistics>true</port-statistics>  <group-statistics>false</group-statistics>  <queue-statistics>true</queue-statistics>  <reassemble-ip-fragments>false</reassemble-ip-fragments>  <block-looping-ports>false</block-looping-ports>  <reserved-port-types>  <type>all</type>  </reserved-port-types>  <group-types>  <type>all</type>  </group-types>  <group-capabilities>  <capability>select-weight</capability>  </group-capabilities>  <action-types>  <type>output</type>  </action-types>  <instruction-types>  <type>apply-actions</type>  <type>write-actions</type>  </instruction-types>  </capabilities> |

### Normative Text

### Yang Specification

|  |
| --- |
| typedef action-type {  description "The types of actions defined in OpenFlow Switch Specification version 1.2.";  type enumeration {  enum output;  enum acopy-ttl-out;  enum copy-ttl-in;  enum set-mpls-ttl;  enum dec-mpls-ttl;  enum push-vlan;  enum pop-vlan;  enum push-mpls;  enum pop-mpls;  enum set-queue;  enum group;  enum set-nw-ttl;  enum dec-nw-ttl;  enum set-field;  }  }  typedef instruction-type {  description "The types of instructions defined in OpenFlow Switch Specification version 1.2.";  type enumeration {  enum apply-actions;  enum clear-actions;  enum write-actions;  enum write-metadata;  enum goto-table;  }  }  grouping openflow-logical-switch-capabilities-grouping {  description "This grouping specifies all properties of an OpenFlow logical switch's capabilities.";  leaf max-buffered-packets {  type uint32;  description "The maximum number of packets the logical switch can buffer when sending packets to the controller using packet-in messages.";  }  leaf max-tables {  type uint8;  description "The number of flow tables supported by the logical switch.";  }  leaf max-ports {  type uint32;  description "The number of flow tables supported by the logical switch.";  }  leaf flow-statistics {  type boolean;  default false;  description "Specifies if the logical switch supports flow statistics.";  }  leaf table-statistics {  type boolean;  default false;  description "Specifies if the logical switch supports table statistics.";  }  leaf port-statistics {  type boolean;  default false;  description "Specifies if the logical switch supports port statistics.";  }  leaf group-statistics {  type boolean;  default false;  description "Specifies if the logical switch supports group statistics.";  }  leaf queue-statistics {  type boolean;  default false;  description "Specifies if the logical switch supports queue statistics.";  }  leaf reassemble-ip-fragments {  type boolean;  default false;  description "Specifies if the logical switch supports reassemble IP fragments.";  }  leaf block-looping-ports {  type boolean;  default false;  description "'true' indicates that a switch protocol outside of OpenFlow, such as 802.1D Spanning Tree, will detect topology loops and block ports to prevent packet loops."  }  container reserved-port-types {  description "Specify generic forwarding actions such as sending to the controller, flooding, or forwarding using non-OpenFlow methods, such as 'normal' switch processing.";  reference "The types of reserved ports are defined in OpenFlow Switch Specification version 1.2.";  leaf-list type {  type enumeration {  enum all;  enum controller;  enum table;  enum inport;  enum any;  enum normal;  enum flood;  }  }  }  container group-types {  description "Specify the group types supported by the logical switch.";  reference "The types of groups are defined in OpenFlow Switch Specification version 1.2.";  leaf-list type {  type enumeration {  enum all;  enum select;  enum indirect;  enum fast-failover;  }  }  }  container group-capabilities {  description "Specify the group capabilities supported by the logical switch.";  reference "The types of group capability are defined in OpenFlow Switch Specification version 1.2.";  leaf-list capability {  type enumeration {  enum select-weight;  enum select-liveness;  enum chaining;  enum chaining-check;  }  }  }  container action-types {  description "Specify the action types supported by the logical switch.";  leaf-list type {  type action-type;  }  }  container instruction-types {  description "Specify the instruction types supported by the logical switch.";  leaf-list type {  type instruction-type;  }  }  } |

## OpenFlow Controller

The OpenFlow Controller class represents an entity that acts as OpenFlow Controller of an OpenFlow Logical Switch. Attributes of the class indicate the role of the controller and parameters of the OpenFlow connection to the controller.

### UML Diagram



Figure : Data Model Diagram for an OpenFlow Controller

### XML Schema

|  |
| --- |
| <xs:complexType name="OFControllerType">  <xs:sequence>  <xs:element name="id"  type="OFConfigID"/>  <xs:element name="role"  type="OFControllerRoleType"/>  <xs:element name="ip-address"  type="inet:ip-prefix"/>  <xs:element name="port"  type="inet:port-number"/>  <xs:element name="local-ip-address"  type="inet:ip-address"/>  <xs:element name="local-port"  type="inet:port-number"/>  <xs:element name="protocol"  type="OFControllerProtocolType"/>  <xs:element name="state"  type="OFControllerOpenFlowStateType"/>  </xs:sequence>  </xs:complexType>  <xs:simpleType name="OFControllerRoleType">  <xs:restriction base="xs:string">  <xs:enumeration value="master"/>  <xs:enumeration value="slave"/>  <xs:enumeration value="equal"/>  </xs:restriction>  </xs:simpleType>  <xs:simpleType name="OFControllerProtocolType">  <xs:restriction base="xs:string">  <xs:enumeration value="tcp"/>  <xs:enumeration value="tls"/>  </xs:restriction>  </xs:simpleType>  <xs:complexType name="OFControllerOpenFlowStateType">  <xs:sequence>  <xs:element name="connection-state"  type="OFControllerConnectionStateType"/>  <xs:element name="current-version"  type="OFOpenFlowVersionType"/>  <xs:element name="supported-versions"  type="OFOpenFlowSupportedVersionsType"/>  </xs:sequence>  </xs:complexType>  <xs:simpleType name="OFControllerConnectionStateType">  <xs:restriction base="xs:string">  <xs:enumeration value="up"/>  <xs:enumeration value="down"/>  </xs:restriction>  </xs:simpleType>  <xs:complexType name="OFOpenFlowSupportedVersionsType">  <xs:sequence>  <xs:element name="version" type="OFOpenFlowVersionType"/>  </xs:sequence>  </xs:complexType>  <xs:simpleType name="OFOpenFlowVersionType">  <xs:restriction base="xs:string">  <xs:enumeration value="1.2"/>  <xs:enumeration value="1.1"/>  <xs:enumeration value="1.0"/>  </xs:restriction>  </xs:simpleType> |

### XML Example

|  |
| --- |
| <controller>  <id>Controller3</id>  <role>master</role>  <ip-address>192.168.2.1/26</ip-address>  <port>6633</port>  <local-ip-address>192.168.2.129</local-ip-address>  <local-port>32768</local-port>  <protocol>tcp</protocol>  <state>  <connection-state>up</connection-state>  <current-version>1.2</current-version>  <supported-versions>  <version>1.2</version>  <version>1.1</version>  </supported-versions>  </state>  </controller> |

### Normative Constraints

An OpenFlow Controller is identified by identifier <id>. The identifier MUST be unique within the context of the OpenFlow Capable Switch. It MUST be persistent across reboots of the OpenFlow Capable Switch.

Element <role> indicates the role of the controller. Semantics of these roles are specified in the OpenFlow 1.2 specification. It is RECOMMENDED that the roles of controllers are not configured by OF-CONFIG 1.0 but determined using the OpenFlow 1.2 protocol. Controllers configured by OF-CONFIG 1.0 SHOULD have the default role “equal”. A role other than “equal” MAY be assigned to a controller. Roles “slave” and “equal” MAY be assigned to multiple controllers. Role “master” MUST NOT be assigned to more than one controller.

Elements <ip-address> and <port> indicate the IP address and the port number of the OpenFlow Controller. The port number is optional. If not present, the default port number 6633 is assumed to be used.

Elements <local-ip-address> and <local-port> indicate the IP address and the port number used by the OpenFlow Logical Switch. Both elements are optional.

Element <protocol> indicates the transport protocol used for the OpenFlow connection. OpenFlow supports two transport protocols, TCP and TLS. If this optional element is not present, TLS is assumed to be used.

Element <state> represents various elements of known state of the OpenFlow Controller. Element <connection-state>represents the administrative state of the OpenFlow connection between the OpenFlow Logical Switch and the OpenFlow Controller. A value of down means that the OpenFlow Logical Switch MUST NOT communicate with the OpenFlow Controller via theOpenFlow protocol. If the value of <connection-state> is set to up, element <current-version>MUST represent the version of the OpenFlow protocol in use between the OpenFlow Logical Switch and the OpenFlow Controller. The element <supported-versions>represents the versions of the OpenFlow protocol that the OpenFlow Controller supports <supported-versions> SHOULD be set to all versions of the OpenFlow protocol supported by the OpenFlow Controller.

The following elements of the OpenFlow Controller can be modified by a NETCONF edit-config request or retrieved by a NETCONF get-config request: <id>, <role>, <ip-address>, <port>, <local-ip-address>, <local-port>, <protocol>, <connection-state>, <current-version>, <supported-versions>.

### YANG Specification

|  |
| --- |
| typedef openflow-version {  type enumeration {  enum "1.0";  enum "1.1";  enum "1.2";  }  description "This enumeration contains the all OpenFlow versions released so far.";  }  grouping openflow-controller-grouping {  description "This grouping specifies all properties of an OpenFlow Logical Switch Controller.";  leaf id {  type inet:uri;  mandatory true;  description "An unique but locally arbitrary identifier that identifies a controller within a OpenFlow Logical Switch and is persistent across reboots of the system.";  }  leaf role {  type enumeration {  enum master;  enum slave;  enum equal;  }  default equal;  description "The predefined role of the controller.";  }  leaf ip-address {  type inet:ip-address;  mandatory true;  description "The IP address of the controller to connect to.";  }  leaf port {  type inet:port-number;  default 6633;  description "The port number at the controller to connect to.";  }  leaf local-ip-address {  type inet:ip-address;  description "This specifies the source IP for packets sent to this controller and overrides the default IP used.";  }  leaf local-port {  type inet:port-number;  default 0;  description "The port number the switch listens on. If 0 the port is chosen dynamically.";  }  leaf protocol {  type enumeration {  enum "tcp";  enum "tls";  }  default "tcp";  description "The protocol used for connecting to the controller.";  }  container state {  description "This container holds connection state information that indicate if the Logical Switch is connected, what versions are supported, and which one is used.";  leaf connection-state {  type up-down-state-type;  description "This object indicates if the Logical Switch is connected to the controller.";  }  leaf current-version {  type openflow-version;  description "This object contains the current OpenFlow version used between Logical Switch and Controller.";  }  leaf-list supported-versions {  type openflow-version;  description "This list of objects contains all the OpenFlow versions supported the controller.";  }  }  } |

## OpenFlow Resource

OpenFlow Resource is a superclass of OpenFlow Port, OpenFlow Queue, Owned Certificate and External Certificate. The superclass contains the identifier attribute that is inherited by all subclasses in addition to their individual identifiers.

### UML Diagram



Figure : Data Model Diagram for an OpenFlow Resource

### XML Schema

|  |
| --- |
| <xs:complexType name="OFResourceType">  <xs:sequence>  <xs:element name="resource-id" type="OFConfigID"/>  </xs:sequence>  </xs:complexType> |

### XML Example

The superclass is not instantiated.

### Normative Constraints

An OpenFlow Resource is identified by identifier <resource-id>. The identifier MUST be unique within the context of the OpenFlow Capable Switch. It MUST be persistent across reboots of the OpenFlow Capable Switch.

### YANG Specification

The base OpenFlow Resource has no specific correspondence in the YANG specification. The <resource-id> property is included in each individual resource.

## OpenFlow Port

The OpenFlow Port is an instance of an OpenFlow resource. It contains a port configuration object, a port state object and a list of port feature objects. While there can’t be more than one instance of the port configuration and the port state, there may be multiple Port Features. The OpenFlow Port is a logical context which represents a port as described in the OpenFlow protocol specification.

### UML Diagram



Figure : Data Model Diagram for an OpenFlow Port

### XML Schema

|  |
| --- |
| <xs:complexType name="OFPortType">  <xs:complexContent>  <xs:extension base="OFResourceType">  <xs:sequence>  <xs:element name="number"  type="xs:unsignedInt"/>  <xs:element name="name"  type="xs:string"/>  <xs:element name="current-rate"  type="xs:unsignedLong"/>  <xs:element name="max-rate"  type="xs:unsignedLong"/>  <xs:element name="configuration"  type="OFPortConfigurationType"/>  <xs:element name="state" type="OFPortStateType"/>  <xs:element name="features"  type="OFPortFeatureMasterList"/>  </xs:sequence>  </xs:extension>  </xs:complexContent>  </xs:complexType>  <xs:complexType name="OFPortFeatureMasterList">  <xs:sequence>  <xs:element name="current"  type="OFPortCurrentFeatureListType"/>  <xs:element name="advertised"  type="OFPortOtherFeatureListType"/>  <xs:element name="supported"  type="OFPortOtherFeatureListType"/>  <xs:element name="advertised-peer"  type="OFPortOtherFeatureListType"/>  </xs:sequence>  </xs:complexType>  <xs:complexType name="OFPortConfigurationType">  <xs:sequence>  <xs:element name="admin-state"  type="OFPortStateOptionsType"/>  <xs:element name="no-receive"  type="xs:boolean"/>  <xs:element name="no-forward"  type="xs:boolean"/>  <xs:element name="no-packet-in"  type="xs:boolean"/>  </xs:sequence>  </xs:complexType>  <xs:complexTypename="OFPortStateType">  <xs:sequence>  <xs:element name="oper-state"  type="OFPortStateOptionsType"/>  <xs:element name="blocked"  type="xs:boolean"/>  <xs:element name="live"  type="xs:boolean"/>  </xs:sequence>  </xs:complexType>  <xs:simpleType name="OFPortStateOptionsType">  <xs:restriction base="xs:string">  <xs:enumeration value="up"/>  <xs:enumeration value="down"/>  </xs:restriction>  </xs:simpleType>  <xs:complexTypename="OFPortCurrentFeatureListType">  <xs:sequence>  <xs:element name="rate"  type="OFPortRateType"/>  <xs:element name="auto-negotiate"  type="OFPortAutoNegotiateType"/>  <xs:element name="medium"  type="OFPortMediumType"/>  <xs:element name="pause"  type="OFPortPauseType" />  </xs:sequence>  </xs:complexType>  <xs:complexType name="OFPortOtherFeatureListType">  <xs:sequence>  <xs:element name="rate"  type="OFPortRateType"  maxOccurs="unbounded"/>  <xs:element name="auto-negotiate"  type="OFPortAutoNegotiateType"/>  <xs:element name="medium"  type="OFPortMediumType"  maxOccurs="unbounded"/>  <xs:element name="pause"  type="OFPortPauseType"/>  </xs:sequence>  </xs:complexType> |

### XML Example

|  |
| --- |
| <port>  <resource-id>Port214748364</resource-id>  <number>214748364</number>  <name>name0</name>  <current-rate>10000</current-rate>  <max-rate>10000</max-rate>  <configuration>  <admin-state>up</admin-state>  <no-receive>false</no-receive>  <no-forward>false</no-forward>  <no-packet-in>false</no-packet-in>  </configuration>  <state>  <oper-state>up</oper-state>  <blocked>false</blocked>  <live>false</live>  </state>  <features>  <current>  ...  </current>  <advertised>  ...  </advertised>  <supported>  ...  </supported>  <advertised-peer>  ...  </advertised-peer>  </features>  </port> |

### Normative Constraints

An OpenFlow Port is identified by identifier <resource-id> within the context of the OpenFlow Capable Switch and OpenFlow Logical Switches. Element <resource-id> is inherited from superclass OpenFlow Resource.

Element <number>identifies the OpenFlow Port to OpenFlow Controllers. If the OpenFlow Port is associated with an OpenFlow Logical Switch, <number> MUST be unique within the context of the OpenFlow Logical Switch.

Element <name> assists OpenFlow Controllers in identifying OpenFlow Ports. <name> MAY be defined. If the OpenFlow Port is associated with an OpenFlow Logical switch and <name> is defined, <name>MUST be unique within the context of the OpenFlow Logical Switch.

Elements <current-rate> and <max-rate> indicate the current and maximum bit rate of the port. Both values are to be provided in units of kilobit per second (kbps). Those elements are only valid if the element <rate> in the current Port Features has a value of other.

Element <configuration>represents the expected behavior of the port based on explicit configuration.

Element <configuration> contains four further elements: <admin-state>, <no-receive>, <no-forward>, <no-packet-in>.

Element <admin-state> represents the configured link state of the port and MUST be set to either up or down.

Element <no-receive> MUST be set to either true or false. A value of true means the port is not receiving any traffic.

Element <no-forward>MUST be set to either true or false. A value of true means the port is not forwarding any packets.

Element <no-packet-in> MUST be set to either true or false. A value of true means port is not receiving any packets.

Element <state> contains three further elements: <oper-state>, <blocked>, <live>.

Element <oper-state> represents the reported link state of the port and MUST have a value of either up or down.

Element <blocked> MUST have a value of either true or false. A value of true means the port has been blocked from receiving or sending traffic.

Element <live> MUST have a value of either true or false .A value of true means the port is active and sending/receiving packets.

An OpenFlow Port contains a list of OpenFlow Port Features in element <features>which contains four sub-lists represented by elements <current>, <advertised>, <supported>,<advertised-peer>.These four lists MUST contain the features associated with the OpenFlow Port. The specific semantics of feature membership in each of these four sub-lists are defined in the OpenFlow protocol.

The following elements of the OpenFlow Port can be modified by a NETCONF edit-config request or retrieved by a NETCONF get-config request: <resource-id>, <number>, <name>, <admin-state>, <no-receive>, <no-forward>, <no-packet-in>.

### YANG Specification

|  |
| --- |
| grouping openflow-port-resource-grouping {  description "This grouping specifies all properties of a port resource.";  leaf resource-id {  type inet:uri;  description "A unique but locally arbitrary identifier that identifies a port and is persistent across reboots of the system.";  }  leaf number {  type uint64;  config false;  mandatory true;  description "An unique but locally arbitrary number that identifies a port and is persistent across reboots of the system.";  }  leaf name {  type string {  length "1..16";  }  config false;  description "Textual port name to ease identification of the port at the switch.";  }  leaf current-rate {  when "../features/current/rate='other'" {  description "This element is only allowed if the element rate of the current features has value 'other'.";  }  type uint32;  units "kbit/s";  config false;  description "The current rate in kilobit/second if the current rate selector has value 'other'.";  }  leaf max-rate {  when "../features/current/rate='other'" {  description "This element is only allowed if the element rate of the current features has value 'other'.";  }  type uint32;  units "kbit/s";  config false;  description "The maximum rate in kilobit/second if the current rate selector has value 'other'.";  }  container configuration {  leaf admin-state {  type up-down-state-type;  default up;  description "The administrative state of the port.";  }  leaf no-receive {  type boolean;  default false;  description "Specifies if receiving packets is not enabled on the port.";  }  leaf no-forward {  type boolean;  default false;  description "Specifies if forwarding packets is not enabled on that port.";  }  leaf no-packet-in {  type boolean;  default false;  description "Specifies if sending packet-in messages for incoming packets is not enabled on that port.";  }  }  container state {  config false;  leaf oper-state {  type up-down-state-type;  mandatory true;  description "The operational state of the port.";  }  leaf blocked {  type boolean;  mandatory true;  description "tbd";  }  leaf live {  type boolean;  mandatory true;  description "tbd";  }  }  container features {  container current {  uses openflow-port-current-features-grouping;  config false;  description "The features (rates, duplex, etc.) of the port that are currently in use.";  }  container advertised {  uses openflow-port-other-features-grouping;  description "The features (rates, duplex, etc.) of the port that are advertised to the peer port.";  }  container supported {  uses openflow-port-other-features-grouping;  config false;  description "The features (rates, duplex, etc.) of the port that are supported on the port.";  }  container advertised-peer {  uses openflow-port-other-features-grouping;  config false;  description "The features (rates, duplex, etc.) that are currently advertised by the peer port.";  }  }  } |

## OpenFlow Port Feature

OpenFlow Port Features includePort Rate, Port Medium, Port Pause, and Port Auto-Negotiate.The normative semantics of these features are described in the OpenFlow protocol specification.

### UML Diagram



Figure : Data Model Diagram for an OpenFlow Port Feature

### XML Schema

|  |
| --- |
| <xs:simpleType name="OFPortRateType">  <xs:restriction base="xs:string">  <xs:enumeration value="10Mb-HD"/>  <xs:enumeration value="10Mb-FD"/>  <xs:enumeration value="100Mb-HD"/>  <xs:enumeration value="100Mb-FD"/>  <xs:enumeration value="1Gb-HD"/>  <xs:enumeration value="1Gb-FD"/>  <xs:enumeration value="1 Tb"/>  <xs:enumeration value="Other"/>  </xs:restriction>  </xs:simpleType>  <xs:simpleType name="OFPortAutoNegotiateType">  <xs:restriction base="xs:string">  <xs:enumeration value="enabled"/>  <xs:enumeration value="disabled"/>  </xs:restriction>  </xs:simpleType>  <xs:simpleType name="OFPortMediumType">  <xs:restriction base="xs:string">  <xs:enumeration value="copper"/>  <xs:enumeration value="fiber"/>  </xs:restriction>  </xs:simpleType>  <xs:simpleType name="OFPortPauseType">  <xs:restriction base="xs:string">  <xs:enumeration value="unsupported"/>  <xs:enumeration value="symmetric"/>  <xs:enumeration value="asymmetric"/>  </xs:restriction>  </xs:simpleType> |

### XML Example

|  |
| --- |
| <rate>10Mb-FD</rate>  <auto-negotiate>enabled</auto-negotiate>  <medium>copper</medium>  <pause>symmetric</pause> |

### Normative Constraints

The OpenFlow Port has several attributes configurable via OF-CONFIG protocol. The normative semantics of these attributes are described in the OpenFlow protocol.

Element <rate> MUST indicate a valid forwarding rate. The current Port Feature set MUST contain this element exactly once. The other Port Feature sets MAY contain this element more than once. If this element appears more than once in a Port Feature set than the value MUST be unique within the Port Feature set.

Element <auto-negotiate>MUST indicate an administrative state of the forwarding rate auto-negotiation protocol.

Element <medium> MUST indicate a valid physical medium. The current Port Feature set MUST contain this element exactly once. The other Port Feature sets MAY contain this element more than once. If this element appears more than once in a Port Feature set than the value MUST be unique within the Port Feature set.

Element <pause>MUST indicate the flavor of the pause function by indicating either asymmetric or asymmetric.

The following elements in the advertised Port Feature set can be modified by a NETCONF edit-config request or retrieved by a NETCONF get-config request: <rate>, <auto-negotiate>, <medium>, <pause>.

### YANG Specification

|  |
| --- |
| typedef rate-type {  type enumeration {  enum 10Mb-HD;  enum 10Mb-FD;  enum 100Mb-HD;  enum 100Mb-FD;  enum 1Gb-HD;  enum 1Gb-FD;  enum 10Gb;  enum 40Gb;  enum 100Gb;  enum 1Tb;  enum other;  }  description "Type to specify the rate of a port including the duplex transmission feature. Possible rates are 10Mb, 100Mb, 1Gb, 10Gb, 40Gb, 100Gb, 1Tb or other. Rates of 10Mb, 100Mb and 1Gb can support half or full duplex transmission.";  }  grouping openflow-port-current-features-grouping {  description "The current features of a port.";  leaf rate {  type rate-type;  mandatory true;  description "The transmission rate that is currently used.";  }  leaf auto-negotiate {  type boolean;  mandatory true;  description "Specifies if auto-negotiation of transmission parameters was used for the port.";  }  leaf medium {  type enumeration {  enum copper;  enum fiber;  }  mandatory true;  description "The transmission medium used by the port.";  }  leaf pause {  type enumeration {  enum unsupported;  enum symmetric;  enum asymmetric;  }  mandatory true;  description "Specifies if pausing of transmission is supported at all and if yes if it is asymmetric or symmetric.";  }  }  grouping openflow-port-other-features-grouping {  description "The features of a port that are supported or advertised.";  leaf-list rate {  type rate-type;  min-elements 1;  description "The transmission rate that is supported or advertised. Multiple transmissions rates are allowed.";  }  leaf auto-negotiate {  type boolean;  mandatory true;  description "Specifies if auto-negotiation of transmission parameters is enabled for the port.";  }  leaf-list medium {  type enumeration {  enum copper;  enum fiber;  }  min-elements 1;  description "The transmission medium used by the port. Multiple media are allowed.";  }  leaf pause {  type enumeration {  enum unsupported;  enum symmetric;  enum asymmetric;  }  description "Specifies if pausing of transmission is supported at all and if yes if it is asymmetric or symmetric.";  }  } |

## OpenFlow Queue

The OpenFlow Queue is an instance of an OpenFlow resource. It contains list of queue properties. The OpenFlow Queue is a logical context which represents a queue as described in the OpenFlow protocol specification.

### UML Diagram



Figure : Data Model Diagram for an OpenFlow Queue

### XML Schema

|  |
| --- |
| <xs:complexType name="OFQueueType">  <xs:complexContent>  <xs:extension base="OFResourceType">  <xs:sequence maxOccurs="1" minOccurs="1">  <xs:element name="id" type="OFConfigID"/>  <xs:element name="port"  type="OFConfigID"/>  <xs:element name="properties"  type="OFQueuePropertiesType"/>  </xs:sequence>  </xs:extension>  </xs:complexContent>  </xs:complexType>  <xs:complexType name="OFQueuePropertiesType">  <xs:sequence>  <xs:element name="min-rate"  type="OFQueueMinRateType"/>  <xs:element name="max-rate"  type="OFQueueMaxRateType"/>  <xs:element name="experimenter"  type="xs:string"/>  </xs:sequence>  </xs:complexType>  <xs:simpleType name="OFQueueMinRateType">  <xs:restriction base="xs:integer"/>  </xs:simpleType>  <xs:simpleType name="OFQueueMaxRateType">  <xs:restriction base="xs:unsignedLong"/>  </xs:simpleType> |

### XML Example

|  |
| --- |
| <queue>  <resource-id>Queue2</resource-id>  <id>2</id>  <port>4</port>  <properties>  <min-rate>10</min-rate>  <max-rate>500</max-rate>  <experimenter>123498</experimenter>  <experimenter>708</experimenter>  </properties>  </queue> |

### Normative Constraints

An OpenFlow Queue is identified by identifier <resource-id> within the context of the OpenFlow Capable Switch and OpenFlow Logical Switches. Element <resource-id> is inherited from superclass OpenFlow Resource.

Element <id>identifies the OpenFlow Queue to OpenFlow Controllers. If the OpenFlow Queue is associated with a OpenFlow Logical Switch, <id>MUST be unique within the context of the OpenFlow Logical Switch.

Element <port> associates an OpenFlow Queue with an OpenFlow Port. If the OpenFlow Queue is associated with an OpenFlow Logical SwitchS and <port> is non-empty, <port> MUST be set to the value of the <resource-id> of an OpenFlow Port which is associated with the OpenFlow Logical Switch S.

Element <properties> indicates the properties associated with the OpenFlow Queue as defined in the OpenFlow protocol specification. If the OpenFlow Queue is associated with an OpenFlow Logical Switch, <properties>MUST include the properties associated to the OpenFlow Queue. Element <properties> contains three possible elements: <min-rate>, <max-rate>, <experimenter>.

Element <min-rate>MUST indicate the minimum rate of the queue by percentage as an integer representing one tenth of one percent.

Element <max-rate>MUST indicate the minimum rate of the queue by percentage as an integer representing one tenth of one percent.

Element <experimenter>MAY indicate values as defined in the OpenFlow protocol specification.

The following elements of the OpenFlow Port can be modified by a NETCONF edit-config request or retrieved by a NETCONF get-config request: <resource-id>, <id>, <port>, <min-rate>, <max-rate>, <experimenter>.

### YANG Specification

|  |
| --- |
| typedef tenth-of-a-percent {  type uint16 {  range "0..1000";  }  units "1/10 of a percent";  description "This type defines a value in tenth of a percent.";  }  grouping openflow-queue-resource-grouping {  description "This grouping specifies all properties of a queue resource.";  leaf resource-id {  type inet:uri;  description "An unique but locally arbitrary identifier that identifies a queue and is persistent across reboots of the system.";  }  leaf id {  type uint64;  mandatory true;  description "An unique but locally arbitrary number that identifies a queue and is persistent across reboots of the system.";  }  leaf port {  type leafref {  path "/capable-switch/resources/port/resource-id";  }  description "Reference to port resources in the Capable Switch.";  }  container properties {  description "The queue properties currently configured.";  leaf min-rate {  type tenth-of-a-percent;  description "The minimal rate that is reserved for this queue in 1/10 of a percent of the actual rate. If not present a min-rate is not set.";  }  leaf max-rate {  type tenth-of-a-percent;  description "The maximum rate that is reserved for this queue in 1/10 of a percent of the actual rate. If not present the max-rate is not set.";  }  leaf-list experimenter {  type uint32;  description "A list of experimenter identifiers of queue properties used.";  }  }  } |

## External Certificate

Instances of an External Certificate contain a certificate that can be used by an OpenFlow Logical Switch for authenticating a controller when a TLS connection is established.

### UML Diagram

to be done

Figure 12: Data Model Diagram for a Certificate

### XML Schema

|  |
| --- |
| <xs:complexType name="OFExternalCertificateType">  <xs:complexContent>  <xs:extension base="OFResourceType">  <xs:sequence maxOccurs="1" minOccurs="1">  <xs:element name="certificate"  type="OFX509CertificateType"/>  </xs:sequence>  </xs:extension>  </xs:complexContent>  </xs:complexType>  <xs:simpleType name="OFX509CertificateType">  <xs:restriction base="base64Binary"></xs:restriction>  </xs:simpleType> |

### XML Example

|  |
| --- |
| <external-certificate>  <resource-id>ownedCertificate3</resource-id>  <certificate>AEF134F56EDB667DFA4320AEF134F56EDB667DFA4320AEF134F  56EDB667DFA4320AEF134F56EDB667DFA4320AEF134F56EDB667DFA4320  ...  AEF134F56EDB667DFA4320AEF134F56EDB667DFA4320AEF134F56EDB667  DFA4320</certificate>  </external-certificate> |

### Normative Constraints

An External Certificate is identified by identifier <resource-id> within the context of the OpenFlow Capable Switch and OpenFlow Logical Switches. Element <resource-id> is inherited from superclass OpenFlow Resource.

Element <certificate> contains an X.509 certificate in DER format base64 encoded.

### YANG Specification

tbd.

## Owned Certificate

Instances of an Owned Certificate contain a certificate and a private key. It can be used by an OpenFlow Logical Switch for authentication itself to a controller when a TLS connection is established.

### UML Diagram

to be done

Figure 13: Data Model Diagram for a Certificate

### XML Schema

|  |
| --- |
| <xs:complexType name="OFOwnedCertificateType">  <xs:complexContent>  <xs:extension base="OFResourceType">  <xs:sequence maxOccurs="1" minOccurs="1">  <xs:element name="certificate"  type="OFX509CertificateType"/>  <xs:element name="private-key"  type="ds:KeyValue"/>  </xs:sequence>  </xs:extension>  </xs:complexContent>  </xs:complexType> |

### XML Example

|  |
| --- |
| <owned-certificate>  <resource-id>ownedCertificate3</resource-id>  <certificate>AEF134F56EDB667DFA4320AEF134F56EDB667DFA4320AEF134F  56EDB667DFA4320AEF134F56EDB667DFA4320AEF134F56EDB667DFA4320  ...  AEF134F56EDB667DFA4320AEF134F56EDB667DFA4320AEF134F56EDB667  DFA4320</certificate>  <private-key>  <ds:RSAKeyValue>  <ds:Modulus>CE45BAF6730F28CDB53534bC4323A333AAF555444DEED233232  ...  </ds:Modulus>  <ds:Exponent>DFA4320AEF134F56EDB66786230900DFA3C6F4443234901234...  </ds:Exponent>  </private-key>  </owned-certificate> |

### Normative Constraints

An Owned Certificate is identified by identifier <resource-id> within the context of the OpenFlow Capable Switch and OpenFlow Logical Switches. Element <resource-id> is inherited from superclass OpenFlow Resource.

Element <certificate> contains an X.509 certificate in DER format base64 encoded. Element <private-key> contains the private key corresponding to the certificate.

### YANG Specification

tbd.

## OpenFlow Flow Table

The OpenFlow Flow Table is an instance of an OpenFlow resource. It contains list of flow table properties. The OpenFlow flow table is a logical context which represents a flow table as described in the OpenFlow protocol specification.

### UML Diagram

TBD

### XML Schema

|  |
| --- |
| <xs:complexType name="OFFlowTableType">  <xs:complexContent>  <xs:extension base="OFResourceType">  <xs:sequence maxOccurs="1" minOccurs="1">  <xs:element name=”max-entries” type=”xs:integer”/>  <xs:element name="instructions" type="OFFlowTableInstructions"/>  <xs:element name="matches" type="OFFlowTableMatches"/>  <xs:element name="write-actions" type="OFFlowTableWriteActions"/>  <xs:element name="apply-actions" type="OFFlowTableApplyActions"/>  <xs:element name="metadata-match" type="xs:hexBinary"/>  <xs:element name="metadata-write" type="xs:hexBinary"/>  </xs:sequence>  </xs:extension>  </xs:complexContent>  </xs:complexType>  <xs:complexType name="OFFlowTableInstructions">  <xs:sequence minOccurs="1" maxOccurs="unbounded">  <xs:element name="type" type="OFInstructionType"/>  </xs:sequence>  </xs:complexType>  <xs:complexType name="OFFlowTableMatches">  <xs:sequence minOccurs="1" maxOccurs="unbounded">  <xs:element name="type" type="OFMatchType"/>  </xs:sequence>  </xs:complexType>  <xs:complexType name="OFFlowTableWriteActions">  <xs:sequence minOccurs="1" maxOccurs="unbounded">  <xs:element name="type" type="OFActionType"/>  </xs:sequence>  </xs:complexType>  <xs:complexType name="OFFlowTableApplyActions">  <xs:sequence minOccurs="1" maxOccurs="unbounded">  <xs:element name="type" type="OFActionType"/>  </xs:sequence>  </xs:complexType>  <xs:simpleType name="OFMatchType">  <xs:annotation>  <xs:documentation> The open flow match types. See OpenFlow protocol 1.2 section A.2.3.7  </xs:documentation>  </xs:annotation>  <xs:restriction base="xs:string">  <xs:enumeration value="input-port"/>  <xs:enumeration value="physical-input-port"/>  <xs:enumeration value="metadata"/>  <xs:enumeration value="ethernet-dest"/>  <xs:enumeration value="ethernet-src"/>  <xs:enumeration value="ethernet-frame-type"/>  <xs:enumeration value="vlan-id"/>  <xs:enumeration value="vlan-priority"/>  <xs:enumeration value="ip-dscp"/>  <xs:enumeration value="ip-ecn"/>  <xs:enumeration value="ip-protocol"/>  <xs:enumeration value="ipv4-src"/>  <xs:enumeration value="ipv4-dest"/>  <xs:enumeration value="tcp-src"/>  <xs:enumeration value="tcp-dest"/>  <xs:enumeration value="udp-src"/>  <xs:enumeration value="udp-dest"/>  <xs:enumeration value="sctp-src"/>  <xs:enumeration value="sctp-dest"/>  <xs:enumeration value="icmpv4-type"/>  <xs:enumeration value="icmpv4-code"/>  <xs:enumeration value="arp-op"/>  <xs:enumeration value="arp-src-ip-address"/>  <xs:enumeration value="arp-target-ip-address"/>  <xs:enumeration value="arp-src-hardware-address"/>  <xs:enumeration value="arp-target-hardware-address"/>  <xs:enumeration value="ipv6-src"/>  <xs:enumeration value="ipv6-dest"/>  <xs:enumeration value="ipv6-flow-label"/>  <xs:enumeration value="icmpv6-type"/>  <xs:enumeration value="icmpv6-code"/>  <xs:enumeration value="ipv6-nd-target"/>  <xs:enumeration value="ipv6-nd-source-link-layer"/>  <xs:enumeration value="ipv6-nd-target-link-layer"/>  <xs:enumeration value="mpls-label"/>  <xs:enumeration value="mpls-tc"/>  </xs:restriction>  </xs:simpleType> |

### XML Example

|  |
| --- |
| <flow-table>  <resource-id>flowtable1</resource-id>  <instructions>  <type>apply-actions</type>  <type>clear-actions</type>  </instructions>  <matches>  <type>input-port</type>  <type>ethernet-dest</type>  </matches>  <write-actions>  <type>output</type>  <type>pop-mpls</type>  </write-actions>  <apply-actions>  <type>output</type>  <type>set-queue</type>  </apply-actions>  <metadata-match>30</metadata-match>  </flow-table> |

### Normative Constraints

TBD

### YANG Specification

|  |
| --- |
| typedef match-type {  description "The types of matches defined in OpenFlow Switch Specification version 1.2.";  type enumeration {  enum input-port;  enum physical-input-port;  enum metadata;  enum ethernet-dest;  enum ethernet-src;  enum ethernet-frame-type;  enum vlan-id;  enum vlan-priority;  enum ip-dscp;  enum ip-ecn;  enum ip-protocol;  enum ipv4-src;  enum ipv4-dest;  enum tcp-src;  enum tcp-dest;  enum udp-src;  enum udp-dest;  enum sctp-src;  enum sctp-dest;  enum icmpv4-type;  enum icmpv4-code;  enum arp-op;  enum arp-src-ip-address;  enum arp-target-ip-address;  enum arp-src-hardware-address;  enum arp-target-hardware-address;  enum ipv6-src;  enum ipv6-dest;  enum ipv6-flow-label;  enum icmpv6-type;  enum icmpv6-code;  enum ipv6-nd-target;  enum ipv6-nd-source-link-layer;  enum ipv6-nd-target-link-layer;  enum mpls-label;  enum mpls-tc;  }  }  typedef hex-binary {  type binary;  description  "hex binary encoded string";  reference  "http://www.w3.org/TR/2004/REC-xmlschema-2-20041028/datatypes.html#hexBinary";  }  grouping openflow-flow-table-resource-grouping {  description "Representation of an OpenFlow Flow Table Resource.";  leaf resource-id {  type inet:uri;  description "An unique but locally arbitrary identifier that identifies a flow table and is persistent across reboots of the system."  }  Leaf max-entries {  Type uint8;  description "The maximum number of flow entries supported by the flow table."  }  container instructions {  leaf-list type {  type instruction-type;  }  description "The instruction types supported by the flow table."  }  container matches {  leaf-list type {  type match-type;  }  description "The match types supported by the flow table."  }  container write-actions {  leaf-list type {  type action-type;  }  description "The write action types supported by the flow table."  }  container apply-actions {  leaf-list type {  type action-type;  }  description "The apply action types supported by the flow table."  }  leaf metadata-match {  type hex-binary;  description "The bits of metadata the flow table can match."  }  leaf metadata-write {  type hex-binary;  description "The bits of metadata the flow table can write."  }  } |

# Binding to NETCONF

The OF-CONFIG1.0 protocol provides a standard way to modify basic OpenFlow configuration for the operation of an OpenFlow logical switch within the context of an OpenFlow Capable Switch. At the same time, it provides vendors the ability to extend and innovate by providing new and improved configuration capabilities. To achieve these goals, OF-CONFIG1.0 requires that devices supporting OF-CONFIG1.0 MUST implement NETCONF protocol (4) as the transport. This in turn implies as specified by NETCONF specification that OpenFlow Capable Switches supporting OF-CONFIG1.0 must implement SSH as a transport protocol. In addition, the OpenFlow Capable Switches implementing OF-CONFIG1.0 protocol may implement additional transports such as Web Services-Management or something else. Future versions of OF-CONFIG may specify binding to these additional transports.

NETCONF is a stable protocol that has been standardized for several years now. It is widely available on various platforms and achieves the needs for OF-CONFIG1.0. NETCONF defines a set of operations on top of a messaging layer (RPC). Below diagram shows the various layers of NETCONF protocol.



Figure NETCONF Layers and Examples

The OpenFlow capable switches MUST support the schema as defined in this specification as the content layer in the above diagram. The schema currently covers basic configuration elements and will be extended in next versions.

The NETCONF protocol meets the OF-CONFIG 1.0 requirements for communication between an OpenFlow Configuration Point and an OpenFlow switch as listed in Section 6.3. In addition, if future needs of OF-CONFIG are not met by NETCONF protocol, NETCONF is extensible which will allow OF-CONFIG to extend NETCONF for its purpose.

1. It supports TLS as communication transport protocol (directly or with SOAP or BEEP in between) that can be used for providing integrity, privacy, and mutual authentication.
2. All specified transport mappings for NETCONF use TLS or TCP as underlying transport protocol and thus provides reliable transport.
3. The common way to establish a connection with NETCONF is from the Configuration Point (configuration point) to the managed device (switch).
4. The NETCONF standard support reversed configuration setup only if BEEP is used as transport protocol.
5. It supports partial switch configuration to the most fine-grain level.
6. It supports full switch configuration with a single operation.
7. It supports setting of configuration data.
8. It supports the retrieval of configuration data.
9. It supports the retrieval of (non-configuration) status data.
10. It supports creation, modification and deletion of configuration information.
11. It supports returning success codes after completing a configuration operation.
12. It supports support reporting error codes for partially or completely failed configuration requests.
13. It supports sending configuration requests independent of the completion of previous requests. Requests may be queued or processed concurrently at a switch. Each request has a request ID. Success or failure indications can be sent independently of other requests individually for each request ID.
14. It supports transaction capabilities including rollback per operation.
15. With its extension defined in RFC 5277 it supports asynchronous notifications from the managed device (switch) to the Configuration Point (configuration point).
16. It is extensible. New operations can be added and its support can be checked by capability retrieval.
17. It supports reporting its capabilities.

## How Data Model is Bound to Netconf

NetConf uses the XML encoding format for requests and responses. More specifically, it uses RPC-based communication model. It uses the <rpc> and <rpc-reply> elements as frames of NetConf requests and responses. The content elements inside of <rpc> element must conform to the OpenFlow Configuraton XML schemas defined in this specification.

All NetConf base protocol operations can be used to retrieve, configure, copy and delete OpenFlow Configuration data stores. These operations are defined in RFC6241. The commonly used operations are:

* edit-config
* get-config
* copy-config
* delete-config

### edit-config

The <edit-config> operation loads all or part of a specifiedconfiguration to the specified target configuration. If the target configuration does not exist, it will be created. The “operation” attribute of elements in the <config> subtree specifies the type of operations to be performed on the element. NetConf supports “create”, “replace”, “merge” and “delete”. The definition of these operations can be found RFC6241.

#### XML Example: Create a Capable-Switch Configuration

This XML example shows an edit-config operation to create a capable-switch configuration.

|  |
| --- |
| <?xmlversion="1.0" encoding="UTF-8"?>  <rpc message-id="1"  xmlns="urn:ietf:params:xml:ns:netconf:base:1.0">  <edit-config xmlns="urn:ietf:params:xml:ns:netconf:base:1.0">  <target>  <candidate/>  </target>  <default-operation>merge</default-operation>  <test-option>set</test-option>  <config>  <capable-switch  xmlns:nc="urn:ietf:params:xml:ns:netconf:base:1.0"  nc:operation="create"  xmlns="urn:onf:of12:config:yang">  <id>capable-switch-0</id>  <logical-switches>  <switch>  <id>logic-switch-1</id>  <datapath-id>11:11:11:11:11:11:11:11</datapath-id>  <enabled>true</enabled>  <controllers>  <controller>  <id>controller-0</id>  <role>master</role>  <ip-address>192.168.2.1</ip-address>  <port>6633</port>  <protocol>tcp</protocol>  </controller>  </controllers>  </switch>  </logical-switches>  </capable-switch>  </config>  </edit-config>  </rpc>  <rpc-reply message-id="1"  xmlns="urn:ietf:params:xml:ns:netconf:base:1.0">  <ok/>  </rpc-reply> |

#### XML Example: Replace the ip-address Element of Controller

This XML example shows an edit-config operation to replace the ip-address element of controller.

|  |
| --- |
| <?xml version="1.0" encoding="UTF-8"?>  <rpc message-id="1"  xmlns="urn:ietf:params:xml:ns:netconf:base:1.0">  <edit-config xmlns="urn:ietf:params:xml:ns:netconf:base:1.0">  <target>  <candidate/>  </target>  <default-operation>merge</default-operation>  <config>  <capable-switch xmlns="urn:onf:of12:config:yang">  <logical-switches>  <switch>  <id>logic-switch-1</id>  <controllers>  <controller>  <id>controller-0</id>  <ip-address operation="replace">10.0.0.10</ip-address>  </controller>  </controllers>  </switch>  </logical-switches>  </capable-switch>  </config>  </edit-config>  </rpc>  <rpc-reply message-id="1"  xmlns="urn:ietf:params:xml:ns:netconf:base:1.0">  <ok/>  </rpc-reply> |

RPC request must contain the key leave(s)( id element in this case) to uniquely identify the element being operated in the NetConf datastore scope.

### get-config

This operation is to retrieve all or part of a specified configuration. The filter element identifies the portions of the OpenFlow configuration to retrieve. If this element is unspecified, the entire configuration is returned.

#### XML Example: get-config

|  |
| --- |
| <?xml version="1.0" encoding="UTF-8"?>  <rpc message-id="1"  xmlns="urn:ietf:params:xml:ns:netconf:base:1.0">  <get-config xmlns="urn:ietf:params:xml:ns:netconf:base:1.0">  <source>  <running/>  </source>  <filter type="xpath" select="/capable-switch"/>  </get-config>  </rpc>  <rpc-reply message-id="1"  xmlns="urn:ietf:params:xml:ns:netconf:base:1.0">  <data>  <capable-switch xmlns="urn:onf:of12:config:yang">  <id>capable-switch-0</id>  <logical-switches>  <switch>  <id>logic-switch-1</id>  <datapath-id>11:11:11:11:11:11:11:11</datapath-id>  <enabled>true</enabled>  <controllers>  <controller>  <id>controller-0</id>  <role>master</role>  <ip-address>192.168.2.1</ip-address>  <port>6633</port>  <protocol>tcp</protocol>  </controller>  </controllers>  </switch>  </logical-switches>  </capable-switch>  </data>  </rpc-reply> |

### copy-config

This operation creates or replaces an entire configuration datastore with the contents of another complete configuration datastore. If the target datastore exists, it is overwritten. Otherwise, a new one is created, if allowed.

#### XML Example: copy-config

|  |
| --- |
| <?xml version="1.0" encoding="UTF-8"?>  <rpc message-id="1"  xmlns="urn:ietf:params:xml:ns:netconf:base:1.0">  <copy-config>  <target>  <running/>  </target>  <source>  <url>https://mydomain.com/of-config/new-config.xml</url>  </source>  </copy-config>  </rpc>  <rpc-reply message-id="1"  xmlns="urn:ietf:params:xml:ns:netconf:base:1.0">  <ok/>  </rpc-reply> |

### delete-config

This operation deletes a configuration datastore. The <running>configuration datastore cannot be deleted.

#### XML Example: delete-config

|  |
| --- |
| <?xml version="1.0" encoding="UTF-8"?>  <rpc message-id="101"  xmlns="urn:ietf:params:xml:ns:netconf:base:1.0">  <delete-config>  <target>  <startup/>  </target>  </delete-config>  </rpc>  <rpc-reply message-id="1"  xmlns="urn:ietf:params:xml:ns:netconf:base:1.0">  <ok/>  </rpc-reply> |

## RPC error

OpenFlow Configuration uses NetConf <rpc-error> element(s) defined in RFC6241 to report operation failures. The <rpc-error> element(s) are sent in <rpc-reply> messages if an error occurs during the processing of an <rpc> request. The <rpc-reply> MAY contain multiple <rpc-error> elements. The <rpc-error>element includes the following information:

* error-type: Defines the conceptual layer of the error occurred.
* error-tag: contains a string to identifying the error condition.
* error-severity: contains a string to identifying the error severity.
* error-app-tag: contains a string to identifying the data-model-specific or implementation-specific error condition.
* error-path: contains the absolute XPath expression identifying the element path associated to the specific error being reported.
* error-message: contains error description suitable for human display
* error-info: contains data-model-specific error content

Detailed <rpc-error> definitions can be found in RFC 6241. Specific implementation may define implementation-specific error information and messages inside of error-info as sub-elements.

An example of <rpc-error> element in <rpc-reply> message:

|  |
| --- |
| <?xml version="1.0" encoding="UTF-8"?>  <rpc-reply message-id="101"  xmlns="urn:ietf:params:xml:ns:netconf:base:1.0"  <rpc-error>  <error-type>application</error-type>  <error-tag> missing-element</error-tag>  <error-severity>error</error-severity>  <error-message xml:lang="en">  expected key leaf in list  </error-message>  <error-info>  <bad-element>id</bad-element>  <error-number>383</error-number>  </error-info>  </rpc-error>  </rpc-reply> |

1. XMLSchema

|  |
| --- |
| <?xml version="1.0" encoding="UTF-8"?>  <xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema"  xmlns:ds="http://www.w3.org/2000/09/xmldsig#"  elementFormDefault="qualified"  targetNamespace="urn:onf:params:xml:ns:onf:of12:config"  xmlns="urn:onf:params:xml:ns:onf:of12:config"  xmlns:of12-config="urn:onf:params:xml:ns:onf:of12:config"  xmlns:inet="urn:ietf:params:xml:ns:yang:ietf-inet-types">  <xs:import namespace="urn:ietf:params:xml:ns:yang:ietf-inet-types"  schemaLocation="ietf-inet-types.xsd"/>  <xs:element name="capable-switch" type="OFCapableSwitchType">  <xs:annotation>  <xs:documentation>The OpenFlow Capable Switch and its configurationpoints, logical switches and resources available to logicalswitches.  </xs:documentation>  </xs:annotation>  </xs:element>  <xs:simpleType name="OFConfigID">  <xs:restriction base="xs:string"/>  </xs:simpleType>  <xs:complexType name="OFCapableSwitchType">  <xs:annotation>  <xs:documentation>Representation of an OpenFlow Capable Switch.</xs:documentation>  </xs:annotation>  <xs:sequence>  <xs:element name="id" type="OFConfigID">  <xs:annotation>  <xs:documentation>An unique but locally arbitrary identifier that identifies a Capable Switch towards management systems and that is persistent across reboots of the system.  </xs:documentation>  </xs:annotation>  </xs:element>  <xs:element name="configuration-points"  type="OFConfigurationPointListType">  <xs:annotation>  <xs:documentation>The list of all configuration points known to the OpenFlow Capable Switch that may manage it using OF-CONFIG.  </xs:documentation>  </xs:annotation>  </xs:element>  <xs:element name="resources"  type="OFCapableSwitchResourceListType">  <xs:annotation>  <xs:documentation>This element contains lists of all resources of the OpenFlow Capable Switch that can be used by OpenFlow Logical Switches.  </xs:documentation>  </xs:annotation>  </xs:element>  <xs:element name="logical-switches"  type="OFLogicalSwitchListType">  <xs:annotation>  <xs:documentation>List of all OpenFlow Logical Switches available on the OpenFlow Capable Switch.  </xs:documentation>  </xs:annotation>  </xs:element>  </xs:sequence>  </xs:complexType>  <xs:complexType name="OFConfigurationPointListType">  <xs:annotation>  <xs:documentation/>  </xs:annotation>  <xs:sequence>  <xs:element name="configuration-point"  type="OFConfigurationPointType"  maxOccurs="unbounded"/>  </xs:sequence>  </xs:complexType>  <xs:complexType name="OFCapableSwitchResourceListType">  <xs:sequence>  <xs:element name="port" type="OFPortType"  maxOccurs="unbounded"/>  <xs:element name="queue" type="OFQueueType"  maxOccurs="unbounded"/>  <xs:element name="owned-certificate"  type="OFOwnedCertificateType" maxOccurs="unbounded"/>  <xs:element name="external-certificate"  type="OFExternalCertificateType"  maxOccurs="unbounded"/>  <xs:element name="flow-table"  type="OFFlowTableType" maxOccurs="unbounded"/>  </xs:sequence>  </xs:complexType>  <xs:complexType name="OFLogicalSwitchListType">  <xs:sequence>  <xs:element name="logical-switch"  type="OFLogicalSwitchType"  maxOccurs="unbounded"/>  </xs:sequence>  </xs:complexType>  <xs:complexType name="OFConfigurationPointType">  <xs:annotation>  <xs:documentation>Representation of an OpenFlow Configuration Point.  </xs:documentation>  </xs:annotation>  <xs:sequence>  <xs:element name="id" type="OFConfigID">  <xs:annotation>  <xs:documentation>An identifier that identifies a Configuration Point of the OpenFlow Capable Switch.  </xs:documentation>  </xs:annotation>  </xs:element>  <xs:element name="uri" type="inet:uri">  <xs:annotation>  <xs:documentation>A locator of the Configuration Point. This element MAY contain a locator of the configuration point including, for example, an IP address and a port number.  </xs:documentation>  </xs:annotation>  </xs:element>  <xs:element name="protocol"  type="OFConfigurationPointProtocolType">  <xs:annotation>  <xs:documentation>The transport protocol that the Configuration Point uses when communicating via NETCONF with the OpenFlow Capable Switch.  </xs:documentation>  </xs:annotation>  </xs:element>  </xs:sequence>  </xs:complexType>  <xs:simpleType name="OFConfigurationPointProtocolType">  <xs:annotation>  <xs:documentation>The mappings of NETCONF to different transport protocols are defined in RFC 6242 for SSH, RFC 4743 for SOAP, RFC 4744 for BEEP, and RFC 5539 for TLS.  </xs:documentation>  </xs:annotation>  <xs:restriction base="xs:string">  <xs:enumeration value="ssh"/>  <xs:enumeration value="soap"/>  <xs:enumeration value="tls"/>  <xs:enumeration value="beep"/>  </xs:restriction>  </xs:simpleType>  <xs:complexType name="OFLogicalSwitchType">  <xs:annotation>  <xs:documentation>The representation of an OpenFlow Logical Switch  </xs:documentation>  </xs:annotation>  <xs:sequence>  <xs:element name="id" type="OFConfigID">  <xs:annotation>  <xs:documentation>An unique but locally arbitrary identifier that identifies an OpenFlow Logical Switch within an OpenFlow Capable Switch. It is persistent across reboots of the system.  </xs:documentation>  </xs:annotation>  </xs:element>  <xs:elementname="capabilities"  type="OFLogicalSwitchCapabilitiesType">  <xs:annotation>  <xs:documentation>Capability items of logical switch.  </xs:documentation>  </xs:annotation>  </xs:element>  <xs:element name="datapath-id" type="OFConfigID">  <xs:annotation>  <xs:documentation>A unique identifier that identifiers an OpenFlow Logical Switch within the context of an OpenFlow Controller.  </xs:documentation>  </xs:annotation>  </xs:element>  <xs:element name="enabled" type="xs:boolean"/>  <xs:element name="check-controller-certificate"  type="xs:boolean"/>  <xs:element name="lost-connection-behavior"  type="OFLogicalSwitchLostConnnectionBehavior"/>  <xs:element name="controllers" type="OFControllerListType">  <xs:annotation>  <xs:documentation>The list of controllers that are assigned to the OpenFlow Logical Switch.  </xs:documentation>  </xs:annotation>  </xs:element>  <xs:element name="resources"  type="OFLogicalSwitchResourceListType">  <xs:annotation>  <xs:documentation>The list of references to all resources of the OpenFlow Capable Switch that the OpenFlow Logical Switch has exclusive access to.  </xs:documentation>  </xs:annotation>  </xs:element>  </xs:sequence>  </xs:complexType>  <xs:simpleType name="OFLogicalSwitchLostConnnectionBehavior">  <xs:restriction base="xs:string">  <xs:enumeration value="failSecureMode"/>  <xs:enumeration value="failStandaloneMode"/>  </xs:restriction>  </xs:simpleType>  <xs:complexType name="OFControllerListType">  <xs:sequence>  <xs:element name="controller"  type="OFControllerType"  maxOccurs="unbounded"/>  </xs:sequence>  </xs:complexType>  <xs:complexType name="OFLogicalSwitchResourceListType">  <xs:sequence>  <xs:element name="port" type="OFConfigID" maxOccurs="unbounded"/>  <xs:element name="queue" type="OFConfigID" maxOccurs="unbounded"/>  <xs:element name="certificate"  type="OFConfigID" minOccurs=0 maxOccurs="1"/>  <xs:element name="flow-table"  type="OFConfigID"maxOccurs="unbounded"/>  </xs:sequence>  </xs:complexType>  <xs:complexType name="OFLogicalSwitchCapabilitiesType">  <xs:sequence>  <xs:element name="max-buffered-packets" type="xs:integer">  <xs:annotation>  <xs:documentation>The maximum number of packets the switch can buffer when sending packets to the controller using packet-in messages. See OpenFlow protocol 1.2 section A.3.1  </xs:documentation>  </xs:annotation>  </xs:element>  <xs:element name="max-tables" type="xs:integer">  <xs:annotation>  <xs:documentation> The number of flow tables supported by the switch. See OpenFlow protocol 1.2 section A.3.1  </xs:documentation>  </xs:annotation>  </xs:element>  <xs:element name="max-ports" type="xs:integer">  <xs:annotation>  <xs:documentation> The number of ports supported by the switch. See OpenFlow protocol 1.2 section A.3.1  </xs:documentation>  </xs:annotation>  </xs:element>  <xs:element name="flow-statistics" type="xs:boolean">  <xs:annotation>  <xs:documentation> Whether the switch supports flow statistics. See OpenFlow protocol 1.2 section A.3.1  </xs:documentation>  </xs:annotation>  </xs:element>  <xs:element name="table-statistics" type="xs:boolean">  <xs:annotation>  <xs:documentation> Whether the switch supports table statistics. See OpenFlow protocol 1.2 section A.3.1  </xs:documentation>  </xs:annotation>  </xs:element>  <xs:element name="port-statistics" type="xs:boolean">  <xs:annotation>  <xs:documentation>Whether the switch supports port statistics. See OpenFlow protocol 1.2 section A.3.1  </xs:documentation>  </xs:annotation>  </xs:element>  <xs:element name="group-statistics" type="xs:boolean">  <xs:annotation>  <xs:documentation> Whether the switch supports group statistics. See OpenFlow protocol 1.2 section A.3.1  </xs:documentation>  </xs:annotation>  </xs:element>  <xs:element name="queue-statistics" type="xs:boolean">  <xs:annotation>  <xs:documentation>Whether the switch supports queue statistics. See OpenFlow protocol 1.2 section A.3.1  </xs:documentation>  </xs:annotation>  </xs:element>  <xs:element name="reassemble-ip-fragments" type="xs:boolean">  <xs:annotation>  <xs:documentation>Whether the switch supports reassemble IP fragments. See OpenFlow protocol 1.2 section A.3.1  </xs:documentation>  </xs:annotation>  </xs:element>  <xs:element name="block-looping-ports" type="xs:boolean">  <xs:annotation>  <xs:documentation>"true" indicates that a switch protocol outside of OpenFlow, such as 802.1D Spanning Tree, will detect topology loops and block ports to prevent packet loops. See OpenFlow protocol 1.2 section A.3.1  </xs:documentation>  </xs:annotation>  </xs:element>  <xs:element name="reserved-port-types"  type="OFReservedPortTypes">  <xs:annotation>  <xs:documentation>Specify generic forwarding actions such as sending to the controller, ooding, or forwarding using non-OpenFlow methods, such as "normal" switch processing. SeeOpenFlow protocol 1.2 section 4.5.  </xs:documentation>  </xs:annotation>  </xs:element>  <xs:element name="group-types" type="OFGroupTypes">  <xs:annotation>  <xs:documentation>The group types supported by the switch. SeeOpenFlow protocol 1.2 section 5.4.1.  </xs:documentation>  </xs:annotation>  </xs:element>  <xs:element name="group-capabilities" type="OFGroupCapabilities">  <xs:annotation>  <xs:documentation>The group capabilities supported by the switch. SeeOpenFlow protocol 1.2 section A.3.5.9.  </xs:documentation>  </xs:annotation>  </xs:element>  <xs:element name="action-types" type="OFActionTypes">  <xs:annotation>  <xs:documentation>The action types supported by the switch. See OpenFlow protocol 1.2 section 5.9 and A.2.5.  </xs:documentation>  </xs:annotation>  </xs:element>  <xs:element name="instruction-types" type="OFInstructionTypes">  <xs:annotation>  <xs:documentation>The instruction types supported by the switch. See OpenFlow protocol 1.2 section 5.6.  </xs:documentation>  </xs:annotation>  </xs:element>  </xs:sequence>  </xs:complexType>  <xs:complexType name="OFReservedPortTypes">  <xs:sequence minOccurs="1" maxOccurs="unbounded">  <xs:element name="type" type="OFReservedPortType"/>  </xs:sequence>  </xs:complexType>  <xs:simpleType name="OFReservedPortType">  <xs:restriction base="xs:string">  <xs:enumeration value="all"/>  <xs:enumeration value="controller"/>  <xs:enumeration value="table"/>  <xs:enumeration value="inport"/>  <xs:enumeration value="any"/>  <xs:enumeration value="local"/>  <xs:enumeration value="normal"/>  <xs:enumeration value="flood"/>  </xs:restriction>  </xs:simpleType>  <xs:complexType name="OFGroupTypes">  <xs:sequence minOccurs="1" maxOccurs="unbounded">  <xs:element name="type" type="OFGroupType"/>  </xs:sequence>  </xs:complexType>  <xs:simpleType name="OFGroupType">  <xs:restriction base="xs:string">  <xs:enumeration value="all"/>  <xs:enumeration value="select"/>  <xs:enumeration value="indirect"/>  <xs:enumeration value="fast-failover"/>  </xs:restriction>  </xs:simpleType>  <xs:complexType name="OFGroupCapabilities">  <xs:sequence minOccurs="1" maxOccurs="unbounded">  <xs:element name="capability" type="OFGroupCapability"/>  </xs:sequence>  </xs:complexType>  <xs:simpleType name="OFGroupCapability">  <xs:restriction base="xs:string">  <xs:enumeration value="select-weight"/>  <xs:enumeration value="select-liveness"/>  <xs:enumeration value="chaining"/>  <xs:enumeration value="chaining-check"/>  </xs:restriction>  </xs:simpleType>  <xs:complexType name="OFActionTypes">  <xs:sequence minOccurs="1" maxOccurs="unbounded">  <xs:element name="type" type="OFActionType"/>  </xs:sequence>  </xs:complexType>  <xs:simpleType name="OFActionType">  <xs:restriction base="xs:string">  <xs:enumeration value="output"/>  <xs:enumeration value="copy-ttl-out"/>  <xs:enumeration value="copy-ttl-in"/>  <xs:enumeration value="set-mpls-ttl"/>  <xs:enumeration value="dec-mpls-ttl"/>  <xs:enumeration value="push-vlan"/>  <xs:enumeration value="pop-vlan"/>  <xs:enumeration value="push-mpls"/>  <xs:enumeration value="pop-mpls"/>  <xs:enumeration value="set-queue"/>  <xs:enumeration value="group"/>  <xs:enumeration value="set-nw-ttl"/>  <xs:enumeration value="dec-nw-ttl"/>  <xs:enumeration value="pop-mpls"/>  <xs:enumeration value="set-field"/>  </xs:restriction>  </xs:simpleType>  <xs:complexType name="OFInstructionTypes">  <xs:sequence minOccurs="1" maxOccurs="unbounded">  <xs:element name="type" type="OFInstructionType"/>  </xs:sequence>  </xs:complexType>  <xs:simpleType name="OFInstructionType">  <xs:restriction base="xs:string">  <xs:enumeration value="apply-actions"/>  <xs:enumeration value="clear-actions"/>  <xs:enumeration value="write-actions"/>  <xs:enumeration value="write-metadata"/>  <xs:enumeration value="goto-table"/>  </xs:restriction>  </xs:simpleType>  <xs:complexType name="OFControllerType">  <xs:annotation>  <xs:documentation>Representation of an OpenFlow Controller  </xs:documentation>  </xs:annotation>  <xs:sequence>  <xs:element name="id" type="OFConfigID">  <xs:annotation>  <xs:documentation>An unique but locally arbitrary identifier that identifies an OpenFlow Controller within the context of an OpenFlow Capable Switch. It is persistent across reboots of the system.  </xs:documentation>  </xs:annotation>  </xs:element>  <xs:element name="role" type="OFControllerRoleType">  <xs:annotation>  <xs:documentation>The predefined role of the controller.  </xs:documentation>  </xs:annotation>  </xs:element>  <xs:element name="ip-address" type="inet:ip-prefix">  <xs:annotation>  <xs:documentation>The remote IP of the controller to connect to.</xs:documentation>  </xs:annotation>  </xs:element>  <xs:element name="port" type="inet:port-number">  <xs:annotation>  <xs:documentation>The port number the controller listens on.  </xs:documentation>  </xs:annotation>  </xs:element>  <xs:element name="local-ip-address" type="inet:ip-address">  <xs:annotation>  <xs:documentation>This specifies the source IP for packets sent to this controller and overrides the default IP used.  </xs:documentation>  </xs:annotation>  </xs:element>  <xs:element name="local-port" type="inet:port-number">  <xs:annotation>  <xs:documentation>The port number the controller listens on. If 0 the port is chosen dynamically.  </xs:documentation>  </xs:annotation>  </xs:element>  <xs:element name="protocol" type="OFControllerProtocolType">  <xs:annotation>  <xs:documentation>The protocol used for connecting to the controller. Both sides must support the chosen protocol for a successful establishment of a connection.  </xs:documentation>  </xs:annotation>  </xs:element>  <xs:element name="state" type="OFControllerOpenFlowStateType">  <xs:annotation>  <xs:documentation>This element represents the state of the OpenFlow protocol connection to the controller.  </xs:documentation>  </xs:annotation>  </xs:element>  </xs:sequence>  </xs:complexType>  <xs:simpleType name="OFControllerRoleType">  <xs:restriction base="xs:string">  <xs:enumeration value="master"/>  <xs:enumeration value="slave"/>  <xs:enumeration value="equal"/>  </xs:restriction>  </xs:simpleType>  <xs:simpleType name="OFControllerProtocolType">  <xs:restriction base="xs:string">  <xs:enumeration value="tcp"/>  <xs:enumeration value="tls"/>  </xs:restriction>  </xs:simpleType>  <xs:complexType name="OFControllerOpenFlowStateType">  <xs:sequence>  <xs:element name="connection-state"  type="OFControllerConnectionStateType">  <xs:annotation>  <xs:documentation>This element represents the run-time state of the OpenFlow connection to the Controller.  </xs:documentation>  </xs:annotation>  </xs:element>  <xs:element name="current-version" type="OFOpenFlowVersionType">  <xs:annotation>  <xs:documentation>This element denotes the version of OpenFlow that Controller is currently communicating with. It is only relevant when the connection-state element is set to "up".  </xs:documentation>  </xs:annotation>  </xs:element>  <xs:element name="supported-versions"  type="OFOpenFlowSupportedVersionsType">  <xs:annotation>  <xs:documentation>This element denotes all of the versions of the OpenFlow protocol that the controller supports.  </xs:documentation>  </xs:annotation>  </xs:element>  </xs:sequence>  </xs:complexType>  <xs:simpleType name="OFControllerConnectionStateType">  <xs:restriction base="xs:string">  <xs:enumeration value="up"/>  <xs:enumeration value="down"/>  </xs:restriction>  </xs:simpleType>  <xs:complexType name="OFOpenFlowSupportedVersionsType">  <xs:sequence>  <xs:element name="version"  type="OFOpenFlowVersionType"  maxOccurs="unbounded"/>  </xs:sequence>  </xs:complexType>  <xs:simpleType name="OFOpenFlowVersionType">  <xs:restriction base="xs:string">  <xs:enumeration value="1.2"/>  <xs:enumeration value="1.1"/>  <xs:enumeration value="1.0"/>  </xs:restriction>  </xs:simpleType>  <xs:complexType name="OFResourceType">  <xs:annotation>  <xs:documentation>A Base Class for OpenFlow Resources.  </xs:documentation>  </xs:annotation>  <xs:sequence>  <xs:element name="resource-id" type="OFConfigID">  <xs:annotation>  <xs:documentation>An unique but locally arbitrary identifier that identifies a resource within the context of and OpenFlow Capable Switch and is persistent across reboots of the system.  </xs:documentation>  </xs:annotation>  </xs:element>  </xs:sequence>  </xs:complexType>  <xs:complexType name="OFPortType">  <xs:complexContent>  <xs:extension base="OFResourceType">  <xs:sequence>  <xs:element name="number" type="xs:unsignedInt"/>  <xs:element name="name" type="xs:string"/>  <xs:element name="current-rate" type="xs:unsignedLong"/>  <xs:element name="max-rate" type="xs:unsignedLong"/>  <xs:element name="configuration" type="OFPortConfigurationType"/>  <xs:element name="state" type="OFPortStateType"/>  <xs:element name="features" type="OFPortFeatureMasterList"/>  </xs:sequence>  </xs:extension>  </xs:complexContent>  </xs:complexType>  <xs:complexType name="OFPortFeatureMasterList">  <xs:sequence>  <xs:element name="current" type="OFPortCurrentFeatureListType"/>  <xs:element name="advertised" type="OFPortOtherFeatureListType"/>  <xs:element name="supported" type="OFPortOtherFeatureListType"/>  <xs:element name="advertised-peer"  type="OFPortOtherFeatureListType"/>  </xs:sequence>  </xs:complexType>  <xs:complexType name="OFPortConfigurationType">  <xs:sequence>  <xs:element name="admin-state" type="OFPortStateOptionsType"/>  <xs:element name="no-receive" type="xs:boolean"/>  <xs:element name="no-forward" type="xs:boolean"/>  <xs:element name="no-packet-in" type="xs:boolean"/>  </xs:sequence>  </xs:complexType>  <xs:complexType name="OFPortStateType">  <xs:sequence>  <xs:element name="oper-state" type="OFPortStateOptionsType"/>  <xs:element name="blocked" type="xs:boolean"/>  <xs:element name="live" type="xs:boolean"/>  </xs:sequence>  </xs:complexType>  <xs:simpleType name="OFPortStateOptionsType">  <xs:restriction base="xs:string">  <xs:enumeration value="up"/>  <xs:enumeration value="down"/>  </xs:restriction>  </xs:simpleType>  <xs:complexType name="OFPortCurrentFeatureListType">  <xs:sequence>  <xs:element name="rate" type="OFPortRateType"/>  <xs:element name="auto-negotiate" type="OFPortAutoNegotiateType"/>  <xs:element name="medium" type="OFPortMediumType"/>  <xs:element name="pause" type="OFPortPauseType"/>  </xs:sequence>  </xs:complexType>  <xs:complexType name="OFPortOtherFeatureListType">  <xs:sequence>  <xs:element name="rate" type="OFPortRateType"  maxOccurs="unbounded"/>  <xs:element name="auto-negotiate" type="OFPortAutoNegotiateType"/>  <xs:element name="medium" type="OFPortMediumType"  maxOccurs="unbounded"/>  <xs:element name="pause" type="OFPortPauseType"/>  </xs:sequence>  </xs:complexType>  <xs:simpleType name="OFPortRateType">  <xs:restriction base="xs:string">  <xs:enumeration value="10Mb-HD"/>  <xs:enumeration value="10Mb-FD"/>  <xs:enumeration value="100Mb-HD"/>  <xs:enumeration value="100Mb-FD"/>  <xs:enumeration value="1Gb-HD"/>  <xs:enumeration value="1Gb-FD"/>  <xs:enumeration value="1 Tb"/>  <xs:enumeration value="Other"/>  </xs:restriction>  </xs:simpleType>  <xs:simpleType name="OFPortAutoNegotiateType">  <xs:restriction base="xs:string">  <xs:enumeration value="enabled"/>  <xs:enumeration value="disabled"/>  </xs:restriction>  </xs:simpleType>  <xs:simpleType name="OFPortMediumType">  <xs:restriction base="xs:string">  <xs:enumeration value="copper"/>  <xs:enumeration value="fiber"/>  </xs:restriction>  </xs:simpleType>  <xs:simpleType name="OFPortPauseType">  <xs:restriction base="xs:string">  <xs:enumeration value="unsupported"/>  <xs:enumeration value="symmetric"/>  <xs:enumeration value="asymmetric"/>  </xs:restriction>  </xs:simpleType>  <xs:complexType name="OFQueueType">  <xs:complexContent>  <xs:extension base="OFResourceType">  <xs:sequence maxOccurs="1" minOccurs="1">  <xs:element name="id" type="OFConfigID">  <xs:annotation>  <xs:documentation>An unique but locally arbitrary number that identifies a queue within the context of and OpenFlow Logical Switch and is persistent across reboots of the system.  </xs:documentation>  </xs:annotation>  </xs:element>  <xs:element name="port" type="OFConfigID">  <xs:annotation>  <xs:documentation>Port in the context of the same Logical Switch which this Queue is associated with.  </xs:documentation>  </xs:annotation>  </xs:element>  <xs:element name="properties" type="OFQueuePropertiesType">  <xs:annotation>  <xs:documentation>Properties of the Queue.  </xs:documentation>  </xs:annotation>  </xs:element>  </xs:sequence>  </xs:extension>  </xs:complexContent>  </xs:complexType>  <xs:complexType name="OFQueuePropertiesType">  <xs:sequence>  <xs:element name="min-rate" type="OFQueueMinRateType"  maxOccurs="1">  <xs:annotation>  <xs:documentation>The minimal rate that is reserved for this queue in 1/10 of a percent of the actual rate.  </xs:documentation>  </xs:annotation>  </xs:element>  <xs:element name="max-rate" type="OFQueueMaxRateType">  <xs:annotation>  <xs:documentation>The maximum rate that is reserved for this queue in 1/10 of a percent of the actual rate.  </xs:documentation>  </xs:annotation>  </xs:element>  <xs:element maxOccurs="unbounded" name="experimenter"  type="xs:unsignedLong">  <xs:annotation>  <xs:documentation>Experimental Properties</xs:documentation>  </xs:annotation>  </xs:element>  </xs:sequence>  </xs:complexType>  <xs:simpleType name="OFQueueMinRateType">  <xs:restriction base="xs:integer"/>  </xs:simpleType>  <xs:simpleType name="OFQueueMaxRateType">  <xs:restriction base="xs:integer"/>  </xs:simpleType>  <xs:complexType name="OFExternalCertificateType">  <xs:complexContent>  <xs:extension base="OFResourceType">  <xs:sequence maxOccurs="1" minOccurs="1">  <xs:element name="certificate"  type="OFX509CertificateType"/>  </xs:sequence>  </xs:extension>  </xs:complexContent>  </xs:complexType>  <xs:complexType name="OFOwnedCertificateType">  <xs:complexContent>  <xs:extension base="OFResourceType">  <xs:sequence maxOccurs="1" minOccurs="1">  <xs:element name="certificate"  type="OFX509CertificateType"/>  <xs:element name="private-key"  type="ds:KeyValue"/>  </xs:sequence>  </xs:extension>  </xs:complexContent>  </xs:complexType>  <xs:simpleType name="OFX509CertificateType">  <xs:restriction base="base64Binary"></xs:restriction>  </xs:simpleType>  <xs:complexType name="OFFlowTableType">  <xs:complexContent>  <xs:extension base="OFResourceType">  <xs:sequence maxOccurs="1" minOccurs="1">  <xs:element name=”max-entries” type=”xs:integer”/>  <xs:element name="instructions" type="OFFlowTableInstructions"/>  <xs:element name="matches" type="OFFlowTableMatches"/>  <xs:element name="write-actions" type="OFFlowTableWriteActions"/>  <xs:element name="apply-actions" type="OFFlowTableApplyActions"/>  <xs:element name="metadata-match" type="xs:hexBinary"/>  <xs:element name="metadata-write" type="xs:hexBinary"/>  </xs:sequence>  </xs:extension>  </xs:complexContent>  </xs:complexType>  <xs:complexType name="OFFlowTableInstructions">  <xs:sequence minOccurs="1" maxOccurs="unbounded">  <xs:element name="type" type="OFInstructionType"/>  </xs:sequence>  </xs:complexType>  <xs:complexType name="OFFlowTableMatches">  <xs:sequence minOccurs="1" maxOccurs="unbounded">  <xs:element name="type" type="OFMatchType"/>  </xs:sequence>  </xs:complexType>  <xs:complexType name="OFFlowTableWriteActions">  <xs:sequence minOccurs="1" maxOccurs="unbounded">  <xs:element name="type" type="OFActionType"/>  </xs:sequence>  </xs:complexType>  <xs:complexType name="OFFlowTableApplyActions">  <xs:sequence minOccurs="1" maxOccurs="unbounded">  <xs:element name="type" type="OFActionType"/>  </xs:sequence>  </xs:complexType>  <xs:simpleType name="OFMatchType">  <xs:annotation>  <xs:documentation> The open flow match types. See OpenFlow protocol 1.2 section A.2.3.7  </xs:documentation>  </xs:annotation>  <xs:restriction base="xs:string">  <xs:enumeration value="input-port"/>  <xs:enumeration value="physical-input-port"/>  <xs:enumeration value="metadata"/>  <xs:enumeration value="ethernet-dest"/>  <xs:enumeration value="ethernet-src"/>  <xs:enumeration value="ethernet-frame-type"/>  <xs:enumeration value="vlan-id"/>  <xs:enumeration value="vlan-priority"/>  <xs:enumeration value="ip-dscp"/>  <xs:enumeration value="ip-ecn"/>  <xs:enumeration value="ip-protocol"/>  <xs:enumeration value="ipv4-src"/>  <xs:enumeration value="ipv4-dest"/>  <xs:enumeration value="tcp-src"/>  <xs:enumeration value="tcp-dest"/>  <xs:enumeration value="udp-src"/>  <xs:enumeration value="udp-dest"/>  <xs:enumeration value="sctp-src"/>  <xs:enumeration value="sctp-dest"/>  <xs:enumeration value="icmpv4-type"/>  <xs:enumeration value="icmpv4-code"/>  <xs:enumeration value="arp-op"/>  <xs:enumeration value="arp-src-ip-address"/>  <xs:enumeration value="arp-target-ip-address"/>  <xs:enumeration value="arp-src-hardware-address"/>  <xs:enumeration value="arp-target-hardware-address"/>  <xs:enumeration value="ipv6-src"/>  <xs:enumeration value="ipv6-dest"/>  <xs:enumeration value="ipv6-flow-label"/>  <xs:enumeration value="icmpv6-type"/>  <xs:enumeration value="icmpv6-code"/>  <xs:enumeration value="ipv6-nd-target"/>  <xs:enumeration value="ipv6-nd-source-link-layer"/>  <xs:enumeration value="ipv6-nd-target-link-layer"/>  <xs:enumeration value="mpls-label"/>  <xs:enumeration value="mpls-tc"/>  </xs:restriction>  </xs:simpleType>  </xs:schema> |

1. YANG Specification

|  |
| --- |
| module onf-config-10 {  namespace "urn:onf:of12:config:yang";  prefix of12-config;  import ietf-inet-types { prefix inet; }  organization  "ONF Config Management Group";  contact  "tbd";  description  "tbd";  revision 2011-12-07 {  description "First Version";  reference "tbd";  }  /\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*  \* Features  \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/  /\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*  \* Type definitions  \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/  typedef openflow-version {  type enumeration {  enum "1.0";  enum "1.1";  enum "1.2";  }  description "This enumeration contains the all OpenFlow  versions released so far.";  }  typedef datapath-id-type {  type string {  pattern  '[0-9a-fA-F]{2}(:[0-9a-fA-F]{2}){7}';  }  description "The datapath-id type represents an OpenFlow  datapath identifier.";  }  typedef tenth-of-a-percent {  type uint16 {  range "0..1000";  }  units "1/10 of a percent";  description "This type defines a value in tenth of a percent.";  }  typedef up-down-state-type {  type enumeration {  enum up;  enum down;  }  description "Type to specify state information for a port or a connection.";  }  typedef rate-type {  type enumeration {  enum 10Mb-HD;  enum 10Mb-FD;  enum 100Mb-HD;  enum 100Mb-FD;  enum 1Gb-HD;  enum 1Gb-FD;  enum 10Gb;  enum 40Gb;  enum 100Gb;  enum 1Tb;  enum other;  }  description "Type to specify the rate of a port including the duplex transmission feature. Possible rates are 10Mb, 100Mb, 1Gb, 10Gb, 40Gb, 100Gb, 1Tb or other. Rates of 10Mb, 100Mb and 1Gb can support half or full duplex transmission.";  }  typedef action-type {  description "The types of actions defined in OpenFlow Switch Specification version 1.2.";  type enumeration {  enum output;  enum acopy-ttl-out;  enum copy-ttl-in;  enum set-mpls-ttl;  enum dec-mpls-ttl;  enum push-vlan;  enum pop-vlan;  enum push-mpls;  enum pop-mpls;  enum set-queue;  enum group;  enum set-nw-ttl;  enum dec-nw-ttl;  enum set-field;  }  }  typedef instruction-type {  description "The types of instructions defined in OpenFlow Switch Specification version 1.2.";  type enumeration {  enum apply-actions;  enum clear-actions;  enum write-actions;  enum write-metadata;  enum goto-table;  }  }  typedef match-type {  description "The types of matches defined in OpenFlow Switch Specification version 1.2.";  type enumeration {  enum input-port;  enum physical-input-port;  enum metadata;  enum ethernet-dest;  enum ethernet-src;  enum ethernet-frame-type;  enum vlan-id;  enum vlan-priority;  enum ip-dscp;  enum ip-ecn;  enum ip-protocol;  enum ipv4-src;  enum ipv4-dest;  enum tcp-src;  enum tcp-dest;  enum udp-src;  enum udp-dest;  enum sctp-src;  enum sctp-dest;  enum icmpv4-type;  enum icmpv4-code;  enum arp-op;  enum arp-src-ip-address;  enum arp-target-ip-address;  enum arp-src-hardware-address;  enum arp-target-hardware-address;  enum ipv6-src;  enum ipv6-dest;  enum ipv6-flow-label;  enum icmpv6-type;  enum icmpv6-code;  enum ipv6-nd-target;  enum ipv6-nd-source-link-layer;  enum ipv6-nd-target-link-layer;  enum mpls-label;  enum mpls-tc;  }  }  typedef hex-binary {  type binary;  description  "hex binary encoded string";  reference  "http://www.w3.org/TR/2004/REC-xmlschema-2-20041028/datatypes.html#hexBinary";  }  /\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*  \* Groupings  \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/  grouping openflow-configuration-point-grouping {  description "Representation of an OpenFlow Configuration Point.";  leaf id {  type inet:uri;  description "An identifier that identifies a Configuration Point of the OpenFlow Capable Switch.";  }  leaf uri {  type inet:uri;  description "A locator of the Configuration Point. This element MAY contain a locator of the Configuration Point including, for example, anIP address and a port number.";  }  leaf protocol {  type enumeration {  enum "ssh";  enum "soap";  enum "tls";  enum "beep";  }  default "ssh";  description "The transport protocol that the Configuration Point uses when communicating via NETCONF with the OpenFlow Capable Switch.";  reference "The mappings of NETCONF to different transport protocols are defined in RFC 6242 for SSH, RFC 4743 for SOAP, RFC 4744 for BEEP, and RFC 5539 for TLS";  }  }  grouping openflow-logical-switch-grouping {  description "This grouping specifies all properties of an OpenFlow Logical Switch.";  leaf id {  type inet:uri;  mandatory true;  description "An unique but locally arbitrary identifier that identifies a Logical Switch within a Capable Switch and is persistent across reboots of the system.";  }  container capabilities {  description "This container specifies all capability items of an OpenFlow Logical Switch.";  uses openflow-logical-switch-capabilities-grouping;  }  leaf datapath-id {  type datapath-id-type;  mandatory true;  description "The datapath identifier of the Logical Switch that uniquely identifies this Logical Switch in the controller.";  }  leaf enabled {  type boolean;  mandatory true;  description "Specifies if the Logical Switch is enabled.";  }  container controllers {  description "The list of controllers for this Logical switch.";  list controller {  key "id";  unique "id";  description "The list of controllers that are assigned to the OpenFlow Logical Switch.";  uses openflow-controller-grouping;  }  }  container resources {  description "The following lists reference to all resources of the OpenFlow Capable Switch that the OpenFlow Logical Switch has exclusive access to.";  leaf-list port {  type leafref {  path "/capable-switch/resources/port/resource-id";  }  description "The list references to all port resources of the OpenFlow Capable Switch that the OpenFlow Logical Switch has exclusive access to.";  }  leaf-list queue {  type leafref {  path "/capable-switch/resources/queue/resource-id";  }  description "The list references to all queue resources of the OpenFlow Capable Switch that the OpenFlow Logical Switch has exclusive access to.";  }  leaf-list flow-table {  type leafref {  path "/capable-switch/resources/flow-table/resource-id";  }  description "The list references to all flow table resources of the OpenFlow Capable Switch that the OpenFlow Logical Switch has exclusive access to.";  }  }  grouping openflow-logical-switch-capabilities-grouping {  description "This grouping specifies all properties of an OpenFlow logical switch's capabilities.";  leaf max-buffered-packets {  type uint32;  description "The maximum number of packets the logical switch can buffer when sending packets to the controller using packet-in messages.";  }  leaf max-tables {  type uint8;  description "The number of flow tables supported by the logical switch.";  }  leaf max-ports {  type uint32;  description "The number of flow tables supported by the logical switch.";  }  leaf flow-statistics {  type boolean;  default false;  description "Specifies if the logical switch supports flow statistics.";  }  leaf table-statistics {  type boolean;  default false;  description "Specifies if the logical switch supports table statistics.";  }  leaf port-statistics {  type boolean;  default false;  description "Specifies if the logical switch supports port statistics.";  }  leaf group-statistics {  type boolean;  default false;  description "Specifies if the logical switch supports group statistics.";  }  leaf queue-statistics {  type boolean;  default false;  description "Specifies if the logical switch supports queue statistics.";  }  leaf reassemble-ip-fragments {  type boolean;  default false;  description "Specifies if the logical switch supports reassemble IP fragments.";  }  leaf block-looping-ports {  type boolean;  default false;  description "'true' indicates that a switch protocol outside of OpenFlow, such as 802.1D Spanning Tree, will detect topology loops and block ports to prevent packet loops."  }  container reserved-port-types {  description "Specify generic forwarding actions such as sending to the controller, flooding, or forwarding using non-OpenFlow methods, such as 'normal' switch processing.";  reference "The types of reserved ports are defined in OpenFlow Switch Specification version 1.2.";  leaf-list type {  type enumeration {  enum all;  enum controller;  enum table;  enum inport;  enum any;  enum normal;  enum flood;  }  }  }  container group-types {  description "Specify the group types supported by the logical switch.";  reference "The types of groups are defined in OpenFlow Switch Specification version 1.2.";  leaf-list type {  type enumeration {  enum all;  enum select;  enum indirect;  enum fast-failover;  }  }  }  container group-capabilities {  description "Specify the group capabilities supported by the logical switch.";  reference "The types of group capability are defined in OpenFlow Switch Specification version 1.2.";  leaf-list capability {  type enumeration {  enum select-weight;  enum select-liveness;  enum chaining;  enum chaining-check;  }  }  }  container action-types {  description "Specify the action types supported by the logical switch.";  leaf-list type {  type action-type;  }  }  container instruction-types {  description "Specify the instruction types supported by the logical switch.";  leaf-list type {  type instruction-type;  }  }  }  grouping openflow-controller-grouping {  description "This grouping specifies all properties of an OpenFlow Logical Switch Controller.";  leaf id {  type inet:uri;  mandatory true;  description "An unique but locally arbitrary identifier that identifies a controller within a OpenFlow Logical Switch and is persistent across reboots of the system.";  }  leaf role {  type enumeration {  enum master;  enum slave;  enum equal;  }  default equal;  description "The predefined role of the controller.";  }  leaf ip-address {  type inet:ip-address;  mandatory true;  description "The IP address of the controller to connect to.";  }  leaf port {  type inet:port-number;  default 6633;  description "The port number at the controller to connect to.";  }  leaf local-ip-address {  type inet:ip-address;  description "This specifies the source IP for packets sent to this controller and overrides the default IP used.";  }  leaf local-port {  type inet:port-number;  default 0;  description "The port number the switch listens on. If 0 the port is chosen dynamically.";  }  leaf protocol {  type enumeration {  enum "tcp";  enum "tls";  }  default "tcp";  description "The protocol used for connecting to the controller.";  }  container state {  description "This container holds connection state information that indicate if the Logical Switch is connected, what versions are supported, and which one is used.";  leaf connection-state {  type up-down-state-type;  description "This object indicates if the Logical Switch is connected to the controller.";  }  leaf current-version {  type openflow-version;  description "This object contains the current OpenFlow version used between Logical Switch and Controller.";  }  leaf-list supported-versions {  type openflow-version;  description "This list of objects contains all the OpenFlow versions supported the controller.";  }  }  }  grouping openflow-port-resource-grouping {  description "This grouping specifies all properties of a port  resource.";  leaf resource-id {  type inet:uri;  description "A unique but locally arbitrary identifier that identifies a port and is persistent across reboots of the system.";  }  leaf number {  type uint64;  config false;  mandatory true;  description "An unique but locally arbitrary number that identifies a port and is persistent across reboots of the system.";  }  leaf name {  type string {  length "1..16";  }  config false;  description "Textual port name to ease identification of the port at the switch.";  }  leaf current-rate {  when "../features/current/rate='other'" {  description "This element is only allowed if the element rate of the current features has value 'other'.";  }  type uint32;  units "kbit/s";  config false;  description "The current rate in kilobit/second if the current rate selector has value 'other'.";  }  leaf max-rate {  when "../features/current/rate='other'" {  description "This element is only allowed if the element rate of the current features has value 'other'.";  }  type uint32;  units "kbit/s";  config false;  description "The maximum rate in kilobit/second if the current rate selector has value 'other'.";  }  container configuration {  leaf admin-state {  type up-down-state-type;  default up;  description "The administrative state of the port.";  }  leaf no-receive {  type boolean;  default false;  description "Specifies if receiving packets is not enabled on the port.";  }  leaf no-forward {  type boolean;  default false;  description "Specifies if forwarding packets is not enabled on that port.";  }  leaf no-packet-in {  type boolean;  default false;  description "Specifies if sending packet-in messages for coming packets is not enabled on that port.";  }  }  container state {  config false;  leaf oper-state {  type up-down-state-type;  mandatory true;  description "The operational state of the port.";  }  leaf blocked {  type boolean;  mandatory true;  description "tbd";  }  leaf live {  type boolean;  mandatory true;  description "tbd";  }  }  container features {  container current {  uses openflow-port-current-features-grouping;  config false;  description "The features (rates, duplex, etc.) of the port that are currently in use.";  }  container advertised {  uses openflow-port-other-features-grouping;  description "The features (rates, duplex, etc.) of the port that are advertised to the peer port.";  }  container supported {  uses openflow-port-other-features-grouping;  config false;  description "The features (rates, duplex, etc.) of the port that are supported on the port.";  }  container advertised-peer {  uses openflow-port-other-features-grouping;  config false;  description "The features (rates, duplex, etc.) that are currently advertised by the peer port.";  }  }  }  grouping openflow-queue-resource-grouping {  description "This grouping specifies all properties of a queue resource.";  leaf resource-id {  type inet:uri;  description "An unique but locally arbitrary identifier that identifies a queue and is persistent across reboots of the system.";  }  leaf id {  type uint64;  mandatory true;  description "An unique but locally arbitrary number that identifies a queue and is persistent across reboots of the system.";  }  leaf port {  type leafref {  path "/capable-switch/resources/port/resource-id";  }  description "Reference to port resources in the Capable Switch.";  }  container properties {  description "The queue properties currently configured.";  leaf min-rate {  type tenth-of-a-percent;  description "The minimal rate that is reserved for this queue in 1/10 of a percent of the actual rate. If not present a min-rate is not set.";  }  leaf max-rate {  type tenth-of-a-percent;  description "The maximum rate that is reserved for this queue in 1/10 of a percent of the actual rate. If not present the max-rate is not set.";  }  leaf-list experimenter {  type uint32;  description "A list of experimenter identifiers of queue properties used.";  }  }  }  grouping openflow-port-current-features-grouping {  description "The current features of a port.";  leaf rate {  type rate-type;  mandatory true;  description "The transmission rate that is currently used.";  }  leaf auto-negotiate {  type boolean;  mandatory true;  description "Specifies if auto-negotiation of transmission parameters was used for the port.";  }  leaf medium {  type enumeration {  enum copper;  enum fiber;  }  mandatory true;  description "The transmission medium used by the port.";  }  leaf pause {  type enumeration {  enum unsupported;  enum symmetric;  enum asymmetric;  }  mandatory true;  description "Specifies if pausing of transmission is supported at all and if yes if it is asymmetric or symmetric.";  }  }  grouping openflow-port-other-features-grouping {  description "The features of a port that are supported or advertised.";  leaf-list rate {  type rate-type;  min-elements 1;  description "The transmission rate that is supported or advertised. Multiple transmissions rates are allowed.";  }  leaf auto-negotiate {  type boolean;  mandatory true;  description "Specifies if auto-negotiation of transmission parameters is enabled for the port.";  }  leaf-list medium {  type enumeration {  enum copper;  enum fiber;  }  min-elements 1;  description "The transmission medium used by the port.  Multiple media are allowed.";  }  leaf pause {  type enumeration {  enum unsupported;  enum symmetric;  enum asymmetric;  }  description "Specifies if pausing of transmission is supported at all and if yes if it is asymmetric or symmetric.";  }  }  grouping openflow-flow-table-resource-grouping {  description "Representation of an OpenFlow Flow Table Resource.";  leaf resource-id {  type inet:uri;  description "An unique but locally arbitrary identifier that identifies a flow table and is persistent across reboots of the system."  }  Leaf max-entries {  Type uint8;  description "The maximum number of flow entries supported by the flow table."  }  container instructions {  leaf-list type {  type instruction-type;  }  description "The instruction types supported by the flow table."  }  container matches {  leaf-list type {  type match-type;  }  description "The match types supported by the flow table."  }  container write-actions {  leaf-list type {  type action-type;  }  description "The write action types supported by the flow table."  }  container apply-actions {  leaf-list type {  type action-type;  }  description "The apply action types supported by the flow table."  }  leaf metadata-match {  type hex-binary;  description "The bits of metadata the flow table can match."  }  leaf metadata-write {  type hex-binary;  description "The bits of metadata the flow table can write."  }  }  /\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*  \* Main container  \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/  container capable-switch {  description "The OpenFlow Capable Switch containing logical switches, and resources that can be assigned to logical switches.";  leaf id {  type inet:uri;  mandatory true;  description "An unique but locally arbitrary identifier that identifies a Capable Switch towards the management system and is persistent across reboots of the system.";  }  container configuration-points {  list configuration-point {  key "id";  unique "id";  description "The list of all Configuration Points known to the OpenFlow Capable Switch that may manage it using OF-CONFIG.";  uses openflow-configuration-point-grouping;  }  }  container resources {  description "A lists containing all resources of the OpenFlow Capable Switch.";  list port {  must "features/current/rate != 'other' or " +  "(count(current-rate) = 1 and count(max-rate) = 1 and " +  " current-rate > 0 and max-rate > 0)" {  error-message "current-rate and max-rate must be specified and greater than 0 if rate equals 'other'";  description "current-rate and max-rate can only be present if rate = 'other, see corresponding leaf descriptions. If rate = 'other', then both leafs must be set to values greater than zero.";  }  key "resource-id";  unique "resource-id";  description "The list contains all port resources of the OpenFlow Capable Switch.";  uses openflow-port-resource-grouping;  }  list queue {  key "resource-id";  unique "resource-id";  description "The list contains all queue resources of the OpenFlow Capable Switch.";  uses openflow-queue-resource-grouping;  }  list flow-table {  key "resource-id";  unique "resource-id";  description "The list contains all flow table resources of the OpenFlow Capable Switch.";  uses openflow-flow-table-resource-grouping;  }  }  container logical-switches {  description "This element contains all OpenFlow Logical Switches on the OpenFlow Capable Switch.";  list switch {  key "id";  unique "id";  description "The list of all OpenFlow Logical Switches on the OpenFlow Capable Switch.";  uses openflow-logical-switch-grouping;  }  }  }  } |

1. Bibliography

1. *OpenFlow Specification 1.2.* **Open Networking Foundation.** 2011.

2. *OpenFlow: enabling innovation in campus networks.* **McKeown, Nick, et al., et al.** 2008, ACM SIGCOMM Computer Communication Review, pp. 69-74.

3. **Bradner, S.** RFC 2119. *IETF.* [Online] March 1997. http://www.ietf.org/rfc/rfc2119.txt.

4. **Enns, et al., et al.** RFC 6241. *IETF.* [Online] June 2011. http://tools.ietf.org/rfc/rfc6241.txt.

1. Revision History

|  |  |  |  |
| --- | --- | --- | --- |
| **Version** | **Date** | **Author** | **Notes** |
| rev1 | 2/11/12 | Cyorke | Moved final 1.0 document to new template |
| rev2 | 3/4/12 | Chuan | Added sections 8.1 and 8.2 |
| rev3 | 3/4/12 | Stu | Edited the UML diagram and updated the XML for 7.3 |
| rev4 | 3/4/12 | Carl | Accepted Stu’s changes and fixed formatting in 7.3 |
| rev4b | 3/6/12 | Stu | Updated 7.3.2 AND 7.3.3Added 7.4 Logical Switch Capabilities. |
| rev5 | 3/6/12 | Juergen | Integrate configuration of certificates forTLS authentication between logical switch and controller. Updated textual descriptions, XML schemas, normative text, XML examples, XML schema in section 7 as well as in Appendix A. |
| rev6 | 3/6/12 | Carl | Accepted and formatted changes by Stu and Juergen. |
| rev7 | 3/12/12 | Thomas | Updated the XML schema for certificates |
| rev8 | 3/13/13 | Chuan | Added section 7.12. Moved all flow table capability items to flow table object. Updated all related XML schema and YANG models. |
| rev9 | 3/18/12 | Carl | Accepted and formatted changes by Chuan. |

1. Considerations for Next or Future Releases

|  |  |  |
| --- | --- | --- |
| **ID** | **Description** | **Priority** |
| F-0001 | Multiple OpenFlow controllers associated with a single OpenFlow capable switch. | P0 |
| F-0002 | Adding additional configuration of queue related attributes beyond what is described in OF 1.1 Section A.2.2 |  |
| F-0003 | OpenFlow Controller configuration and monitoring |  |
| F-0004 | bootstrap/auto-discovery/auto-associate of OpenFlow Capable Switches and the OpenFlow Manager |  |

1. Appendix E: ToDo List

|  |  |  |  |
| --- | --- | --- | --- |
| **Date** | **Owner** | **Description** | **Status** |
| 10/6/11 | Stu | Change Port Configuration and Port State to Enums | Done |
| 10/6/11 | Stu | Add text strongly suggesting that OpenFlow Capable Switches avoid changing anything within the OpenFlow operational context with mechanisms other than OF-CONFIG 1.2. This is an especially strong recommendation with respect to changes within the OpenFlow Logical Switch context | Done |
| 10/6/11 | Stu | Add a requirement that the OF Manger be notified of changes to the OpenFlow operational context of an OpenFlow Capable Switch which occur via a channel other than OF-CONFIG 1.3, | Postponed |
| 10/6/11 | Stu | Develop an alternative model to consider for OpenFlow Port Feature Lists where there is one list and each feature is “tagged” with Current, Supported, Advertised, properties | Done  Expressed in XML Schema |
| 10/6/11 | Stu | Work out details for consideration of using URI as the universal identifier scheme including the enumeration identifiers of OpenFlow Resource | Done  (use URNs) |
| 10/6/11 | Stu | Update top level UML model to reflect the need for a backup/failover OpenFlow Controller associated with an OpenFlow Logical Switch | Done |
| 10/6/11 | Stu | Updated Use-cases to reflect that an OpenFlow Capable Switch can be configured to associate with a known OpenFlow Manager and an OpenFlow Manager can be configured to associate with a known OpenFlow Capable Switch. | Done |
| 10/16/11 | Stu | Support F-0001 in Schema | Done |
|  |  |  |  |
| 10/20/11 | Stu | Propose a field or entry point to bind Resources to a context outside of the Logical Switch | Postpone |
| 10/20/11 | Stu | Add the controller roles fields to the controller UML | Done |
| 10/31/11 | Stu | Add protocol and local ip fields to the controller class | Done |
| 10/31/11 | Stu | Add schema XSD | Done |
| 10/31/11 |  | Add language binding to NETCONF | Done |
| 10/31/11 | Thomas | Change ids to URNs and to remove emergency behavior from XSD (schema) | Done |
| 10/31/11 | Thomas | XSD (Schema) change: make the logical switch list a child of capable switch | Done |
| 11/3/11 | Stu | Add (Master, Slave, Equal) for the Controller Role Enum | Done |
| 11/3/11 | Stu | Add new XML schema with URNs to spec | Done |
| 11/3/11 | Thomas | Obtain URN namespace for OFConfig |  |
| 11/3/11 | Thomas | Make sure XSD has both Resourced ID and Controller scoped identifier (e.g. port num, queue id, etc.) | Done |
| 11/3/11 | Stu | Move Schema to the appendix | Done |
| 11/7/11 | Deepak | Motivation and Scope | Done |
| 11/7/11 | Juergen | Add normative constraints with IETF normative language (MUSTs) with referencing the XML | Done |
| 11/7/11 | Stu | Delete Meters | Done |
| 11/7/11 | All | Add descriptive language to the data model detail | Done |
| 11/7/11 | Stu | Add references | Done |
| 11/7/11 | Thomas | Add YANG examples in the data model detail and add the complete YANG module in Appendix B | Done |
| 11/7/11 | Stu | Bring the XML and UML in alignment for Port Features | Done |
| 11/9/11 | Stu | Double Check figure numbers | Done |
| 11/9/11 | Stu | Update XSD and XML for port feature | Done |
| 11/20/11 | Stu | Add references for UML and XML, YANG Schema Standards | Done |
| 11/20/11 | Stu | YANG formatted | Done in 13R5 |
| 12/5/11 | Stu | set min-occurs to 0 for non-identifying elements. |  |
| 12/5/11 | Stu | (re)add operations section which gives XML examples of NETCONF's edit-config, get-config, etc. | Maybe Juergen will take it |
| 12/5/11 | Thomas/Juergen | add example (non-normative) use of embedded "port number" in the resource-id of ports | Done |
| 12/5/11 | Stu | Clarify configuration point semantics (Chris sent out use-case) |  |
| 12/15/11 | Juergen | Add Port state configuration requirement even though it is not explicitly called out in openflow 1.2 spec | Done |
| 12/15/11 | Juergen | 7.3 - 2: change to must support connection from switch -> OFCP and should support from OFCP-> switch | Done |
| 12/15/11 | Any | Reword 7.3 – 4 and 5 to refer to OFCP rather than manager |  |
| 12/15/11 | Juergen | Add requirement around error and status for configuration | Done |
| 12/15/11 | Juergen | Add requirement around reliable transport | Done |
| 12/15/11 | Juergen | Decide on semantics and requirements of async operations and notifications | Done |
| 12/20/11 | Juergen | Bulk operations | Done |
| 12/7/11 | Stu | Renumber of-config 1.2 to of-config 1.0. This makes it clear that of-config is a separate spec from openflow and enables independent evolution of of-config both in terms of timelines and content/scope. | Done |
| 12/7/11 | Deepak | Of-config 1.0 is focused on configuring the openflow datapath. Motivation and some other sections do not make it clear and create confusion by referring to broad OAM. This should be clarified. | Done |
| 12/7/11 | Juergen | Beef up the requirements section to include requirements for the protocol as well. These requirements were discussed within the working group early on (in terms of the spreadsheet that Juergen had sent out originally). It would be good to summarize those requirements in the requirement section. | Outdated (first version done) |
| 12/7/11 | Stu | Structure the doc to be top down: Start with the scope of 1.0, then the requirements for 1.0, then the requirements of the protocol for 1.0, which can then lead into stating clearly why Netconf was selected (and its pros and cons). | Done |
| 12/7/11 | Stu | Rename OF-Manager to OpenFlow Configuration Endpoint (OFCP). This will help clarify the scope of of-config 1.0 and avoids confusion that of-config 1.0 solves broad OAM problem as the term manager may imply. | Done |
| 12/23/11 | Stu | Root cause example indenting loss |  |
| 12/23/11 | Stu | Add NETCONF get-config, edit-config examples |  |
| 12/23/11 | Stu | Resolve the two experiments values in queue |  |
| 1/6/12 | Carl | Add new paragraph tags for formatting XML. Put in changes from Deepak and Michael. |  |
| 1/9/12 | Carl | Reformat XML, fix new paragraph tags. |  |

1. Notes From The Board And The Tag

|  |  |  |  |
| --- | --- | --- | --- |
| **DATE** | **AUTHOR** | **SUBJECT** | **NOTE** |
| 8/16/11 | Dan Pitt | East-West Interface | There is one component of this otherwise thorough plan that I think should be postponed: controller configuration and monitoring. The urgent need now is for switch configuration and management, and there is considerable work to do there. No one has anywhere near enough experience with controllers to know what they should even include, so standardizing them would, I fear, stifle innovation. I am also quite sure the Board would not approve any group's charter to standardize the controller or its northbound interfaces anytime soon. Personally, I want to see controller companies succeed, so let's not tie their hands when they should be experimenting with technologies, business models, and users. |