OPEN NETWORKING
FOUNDATION

OpenFlow Management and Configuration Protocol (OF-Config 1.1)

Version 1.1

June 25, 2012

ONF TS-005

ONF Document Type: OpenFlow Config
ONF Document Name: of-config-1.1

Disclaimer

Contents

1 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 1.3 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 1.1 of OF-CONFIG.

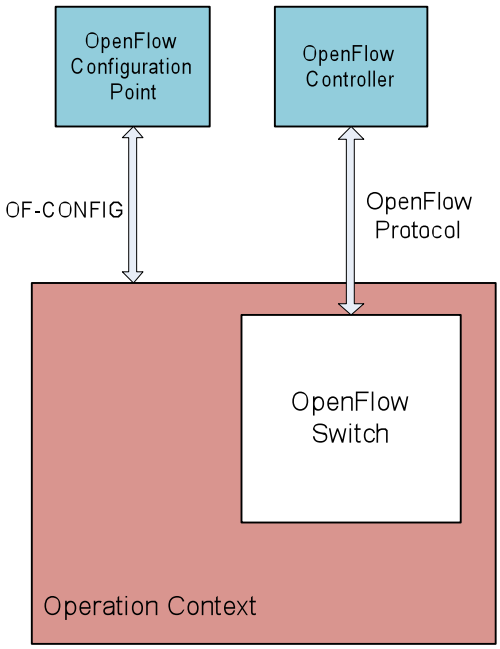

Figure 1: An OpenFlow Configuration Point communicates with an operational context which is capable of supporting an OpenFlow Switch using the OpenFlow Configuration and Management Protocol (OF-CONFIG)

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 logical switch described in this document via OF-CONFIG and limit changes to the OpenFlow logical switche 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.

2 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. An example is building forwarding

tables and deciding forwarding actions which are done via Openflow protocol while enabling/disabling a port does not need to be done at the timescale of a flow and hence, is done via OF-Config protocol.

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 1.1 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 1.1 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 1.1.

Figure 2 shows the basic abstractions detailed in OF-CONFIG 1.1 and the lines indicate that the OpenFlow Configuration Points and OpenFlow Capable Switches communicate via OF-OFCONFIG. The configuration settings then take effect on targeted logical switch(es). OpenFlow Controllers and OpenFlow Logical Switches (i.e. datapaths) communicate via OpenFlow.

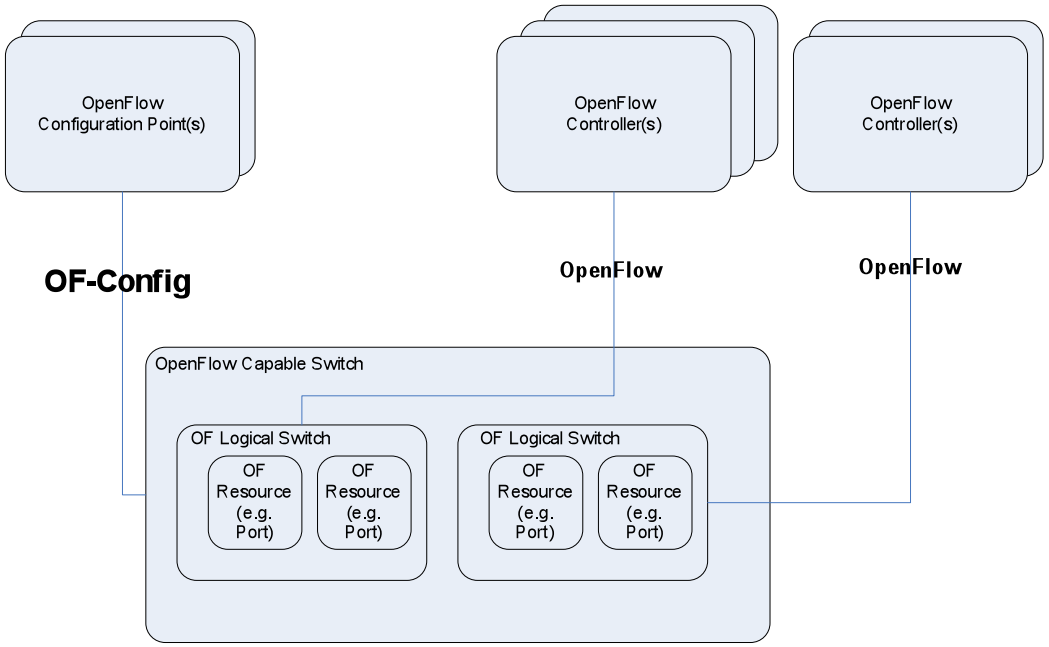

Figure 2: Relationship between components defined in this specification, the OF-CONFIG protocol and the OpenFlow protocol

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.

3 Scope

OF-CONFIG 1.1 is focused on the following functions needed to configure an OpenFlow 1.3 (OFv1.2) datapath:

- 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)
- Configuration of ceritificates for secure communication between the OpenFlow Logical Switches and OpenFlow Controllers
- Discovery of capabilities of an OpenFlow Logical Switch
- Configuration of a small set of tunnel types such as IP-in-GRE, NV-GRE, VxLAN

While limited in scope, OF-CONFIG 1.1 lays the foundation on top of which various automated and more advanced configurations will be possible in future revisions.Switch discovery, topology discovery, capability configuration, event triggers, versioning, 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 1.1 protocol. These may be included in future versions.

4 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).

5 Terms

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

5.1 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.

5.2 OpenFlow Configuration Point

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

5.3 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).

5.4 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.

5.4.1 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.

5.4.2 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. An Openflow Port may map to a physical port on a physical switch or a logical port on a physical or virtual switch.

5.5 OpenFlow Controller

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

6 Requirements

This section describes requirements for the design of OF-CONFIG 1.1.

6.1 Requirements from the OpenFlow 1.3 Protocol Specification

The specification of version 1.3 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 1.1 enables other configuration of OpenFlow switches. The specification of OF-CONFIG 1.1 is written with extensibility in mind. This includes versioning and backward compatibility. Whereas the current specification does not explicitly uses versioning, those features are inherent capabilities of the chosen protocol. In a future version OF-CONFIG will make use of these features and will include versioning, which in turn will enable backward compatibility.

6.1.1 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 1.1 must provide means for configuring these parameters.

6.1.2 Multiple Controllers

Section 6.3 of (1) discusses how a switch deals with multiple controllers simultaneously. This implicitly requires OF-CONFIG 1.1 to provide means for configuring multiple instances of the parameter set listed in 6.1.1 for specifying the connection setup to multiple controllers.

6.1.3 OpenFlow Logical Switches

The OpenFlow 1.3 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.

6.1.4 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.

6.1.5 Encryption

Section 6.5 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.

6.1.6 Queues

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

- min-rate
- max-rate
- experimenter

OF-CONFIG 1.1 must provide means for configuring these parameters.

6.1.7 Ports

The OpenFlow protocol already contains methods to configure a limited amount of port parameters 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 arequirement for OF-CONFIG 1.1. Section A.3.4.3 of (1) defines the following parameters for port configuration:

- no-recveive
- no-forward
- no-packetin
- admin-state

OF-CONFIG 1.1 must provide means for configuring these parameters.

Also defined in Section A.2.1 of the OpenFlow protocol specification (1) 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 1.1 must provide means for configuring these advertised features.

Section 4.4 of (1) defines logical ports that are higher level abstratcions and that may include encapsulation. In addition, logical ports support passing of meta data to the controller. OF-CONFIG 1.1 must support the configuration of these logical ports. However, the configuration of logical ports in OF-Config 1.1 is limited to a small number of tunnels (specifically to IPinGRE, VxLAN and NVGRE) that may be used in datacenter scenarios like network virtualization. Future versions of OF-Config will support configuration of additional types of tunnels.

6.1.8 Capability Discovery

OpenFlow 1.3 describes the various capabilities that an OpenFlow Logical Switch may implement eg there are several actions in OpenFlow 1.3 that are optional. While configuration of these capabilities is outside the scope of OF-CONFIG 1.1, it supports discovery of these capabilities. It is assumed that capabilities have been configured for OpenFlow Logical switches either as part of instantiation of these switches or through some out of band mechanisms.

6.1.9 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.

6.2 Operational Requirements

The OF-CONFIG 1.1 must meet support the following scenarios:

1. OF-CONFIG 1.1 must support an OpenFlow Capable Switch being configured by multiple OpenFlow Configuration Points.
2. OF-CONFIG 1.1 must support an OpenFlow Configuration Point managing multiple OpenFlow Capable Switches.

3. OF-CONFIG 1.1 must support an OpenFlow Logical Switch being controlled by multiple OpenFlow Controllers.

4. OF-CONFIG 1.1 must support configuring ports and queues of an OpenFlow Capable Switch that have been assigned to an OpenFlow Logical Switch.

5. OF-CONFIG 1.1 must support discovery of capabilities of an OpenFlow Logical Switch.

6. OF-CONFIG 1.1 must support configuration of tunnels such as IP-in-GRE, NVGRE and VxLan that are represented as logical ports of an OpenFlow Logical Switch.

6.3 Requirements for the Switch Management Protocol

OF-CONFIG 1.1 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. Note that these requirements are a superset of the requirements that may be needed for the limited scope of configuration specified in this specifications. The intent for the below requirements is to future proof the protocol choice so that we are able to address the future scenarios without having to modify the protocol choice itself. 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. An example may be, even though this scenario is out of scope for OF-CONFIG 1.1, is if an administrator changes a configuration out of band, the switch may need to provide an appropriate notification to the OFCP.

16. The protocol should be extensible.

17. The protocol should support reporting its capabilities.

7 Data Model

This section specifies the data model for OF-CONFIG 1.1. 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 schema extracted 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 class extending the XML schema by semantic specifications,

5. a portion of a YANG (9) module extracted 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 1.1 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.

One of the desing goals of the model is efficient and clear encding of switch configurations in XML. Human readability is a strong feature of XML. But since the XML schema will mainly be created and parsed by the protocol entity, the ease of encoding and parsing was preferred over readability. This implies that in case of a trade-off between cleanness and simplicity of the XML-based configuration and simplicity of the XML schema, usually cleanness and simplicity of the XML-based configuration has been preferred.

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.

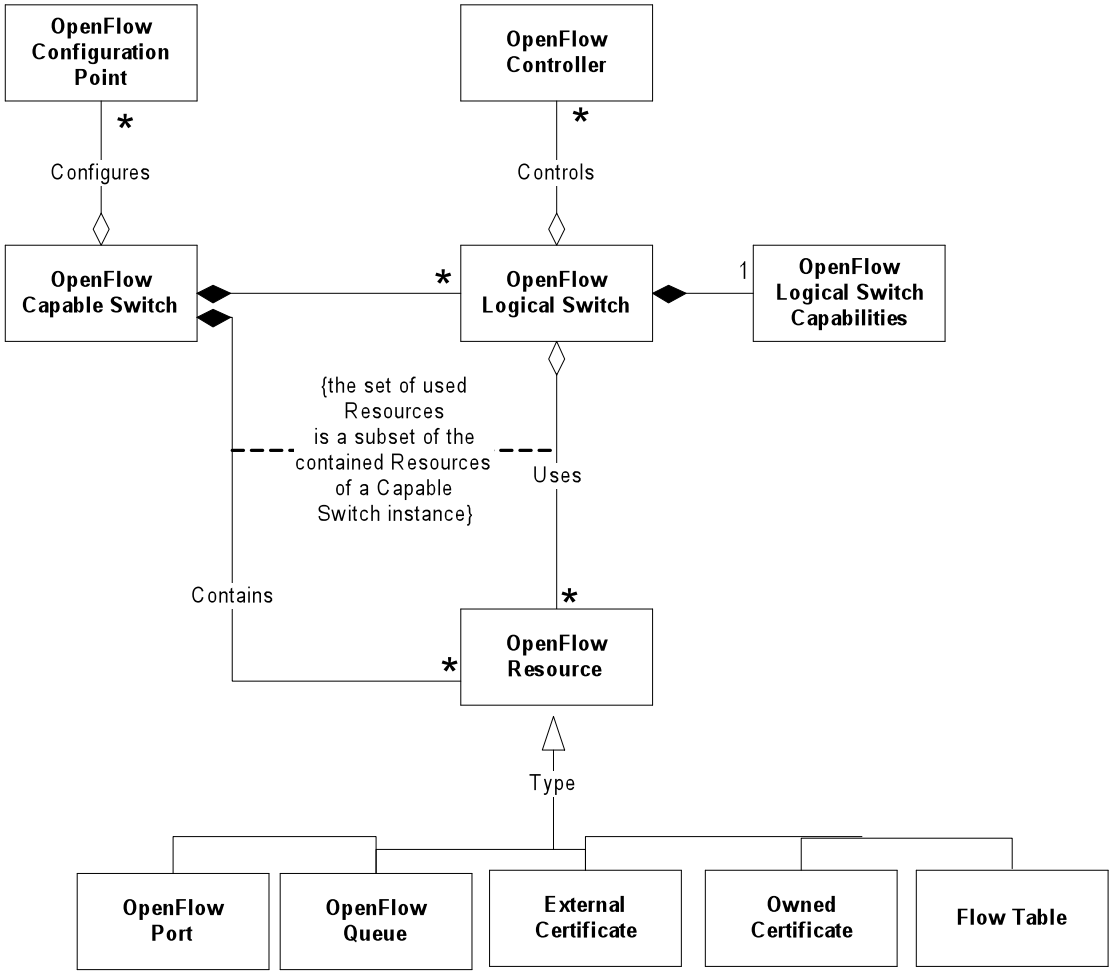

Figure 3: UML Class Diagram for OF-CONFIG Data Model

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 1.1, several types of resources are included in the model: OpenFlow Ports, OpenFlow Queues, External Certificate, Owned Certificate and Flow Table. 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 v1.1.

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.

7.1 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.

7.1.1 UML Diagram

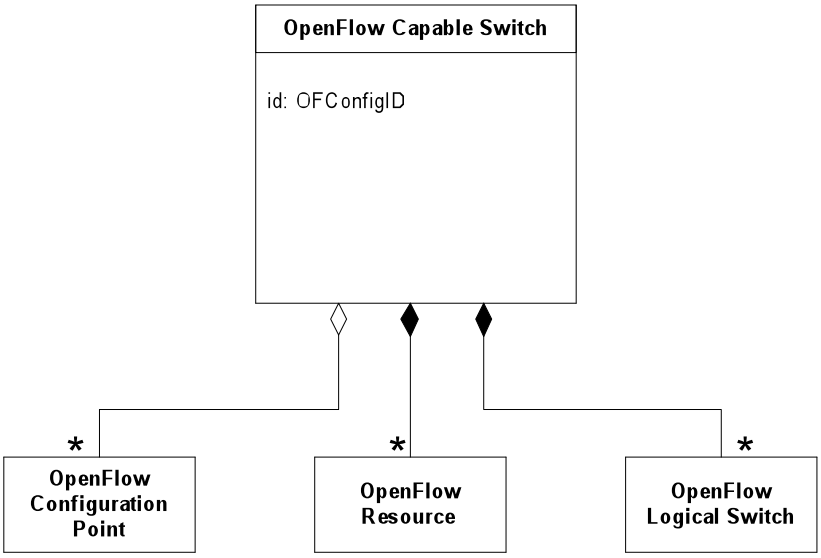

Figure 4: Data Model Diagram for OpenFlow Capable Switch

7.1.2 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>
```

7.1.3 XML Example

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

7.1.4 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.

7.1.5 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;
     }
   }
}
```


7.2 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. Nevertheless SSH is used as a default connection protocol because connection initiation by the switch is optional.

7.2.1 UML Diagram

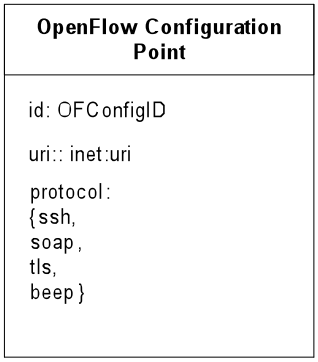

Figure 5: Data Model Diagram for an OpenFlow Configuration Point

7.2.2 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>
```

7.2.3 XML Example

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

7.2.4 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 a connection is established between an OpenFlow Capable Switch and a Configuration Point the switch must store the connection information in an instance of the Configuration Point class. If such an instance does not exist, the OpenFlow Capable Switch MUST create an instance where it then stores the 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>`.

7.2.5 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";
  }
}
```

7.3 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

7.3.1 UML Diagram

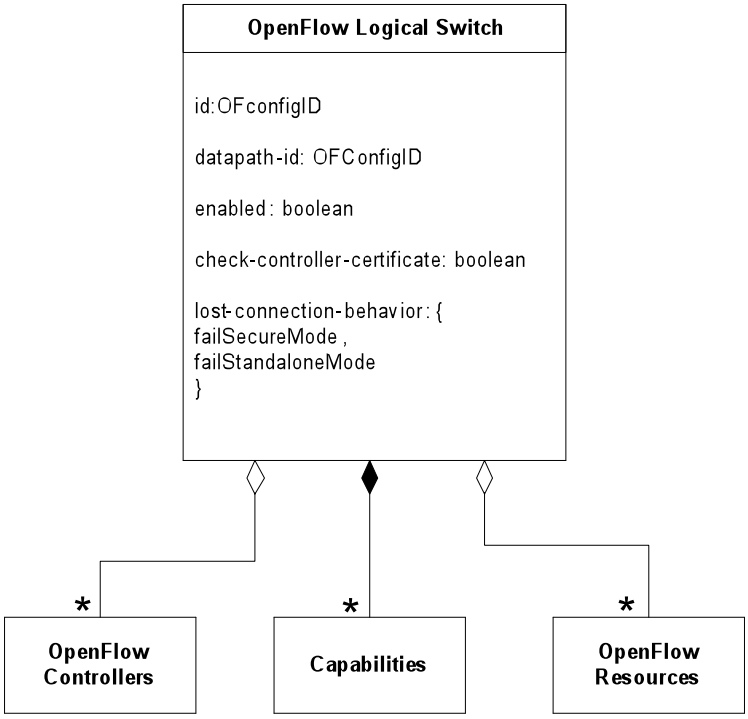

Figure 6: Data Model Diagram for an OpenFlow Logical Switch

7.3.2 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>
```

7.3.3 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>
```

7.3.4 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.

Element <capabilities> contains the capability items the OpenFlow Logical Switch MAY implement. Configuration of these capability items are out of scope of OF-CONFIG1.1. These OpenFlow Logical Switch items MAY be discovered by the configuration point using a NETCONF get-config request. Capability item definition details are included in section 7.4.

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 element <resource-id> of a resource of the OpenFlow Capable Switch.

Any <port>, <queue> or <flow-table> resource identified in the <resources> list of an OpenFlow 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.

7.3.5 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 certificate {
      type leafref {
        path "/capable-switch/resources/owned-certificate/resource-id";
      }
      description "The reference to the owned certificate in
        the OpenFlow Capable Switch that the OpenFlow Logical
        Switch used to identify itself.";
    }
    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.";
    }
  }
}
```

7.4 Logical Switch Capabilities

7.4.1 UML Diagram

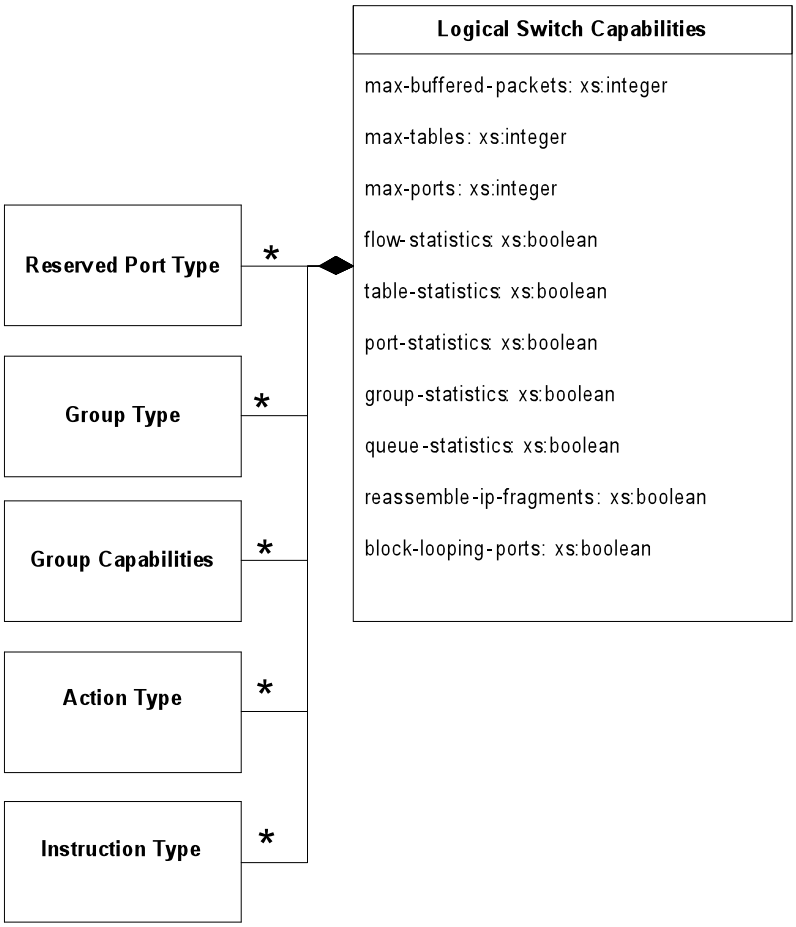

Figure 7: Data Model Diagram for an OpenFlow Logical Switch Capabilities

7.4.2 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, flooding, 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>
```

7.4.3 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>
```

7.4.4 Normative Constraints

Element `<capabilities>` contains the capability items the OpenFlow Logical Switch MAY implement. Configuration of these capability items are out of scope of OF-CONFIG1.1. It is assumed that capabilities have been configured for OpenFlow Logical switches either as part of instantiation of these switches or through some out of band mechanisms.These OpenFlow Logical Switch items can be discovered by the configuration point using a NETCONF get-config request.

Element `<max-buffered-packets>` denotes the maximum number of packets the logical switch can buffer when sending packets to the controller using packet-in messages.

Element `<max-tables>` denotes the maximum number of flow tables supported by the logical switch.

Element `<max-ports>` denotes the maximum number of ports supported by the logical switch.

Element `<flow-statistics>` indicates whether the logical switch supports flow statistics functionality.

Element `<table-statistics>` indicates whether the logical switch supports table statistics functionality.

Element `<port-statistics>` indicates whether the logical switch supports port statistics functionality.

Element `<group-statistics>` indicates whether the logical switch supports group statistics functionality.

Element `<queue-statistics>` indicates whether the logical switch supports queue statistics functionality.

Element `<reassemble-ip-fragments>` indicates whether the logical switch support reassemble IP fragments functionality.

Element `<block-looping-ports>` indicates whether a switch protocol outside of OpenFlow, such as 802.1D Spanning Tree, will detect topology loops and block ports to prevent packet loops.

Element `<reserved-port-types>` denotes generic forwarding actions such as sending to the controller, flooding, or forwarding using non-OpenFlow methods, such as "normal" switch processing.

Element `<group-types>` denotes the group types supported by the OpenFlow logical switch.

Element `<group-capabilities>` denotes the group capabilities supported by the OpenFlow logical switch.

Element `<action-types>` denotes the action types supported by the OpenFlow logical switch.

Element <instruction-types> denotes the instruction types supported by the OpenFlow logical switch.

Detailed definitions of these capability items can be found in OpenFlow Switch Specification 1.2[1].

7.4.5 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;
     }
   }
}
```

7.5 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.

7.5.1 UML Diagram

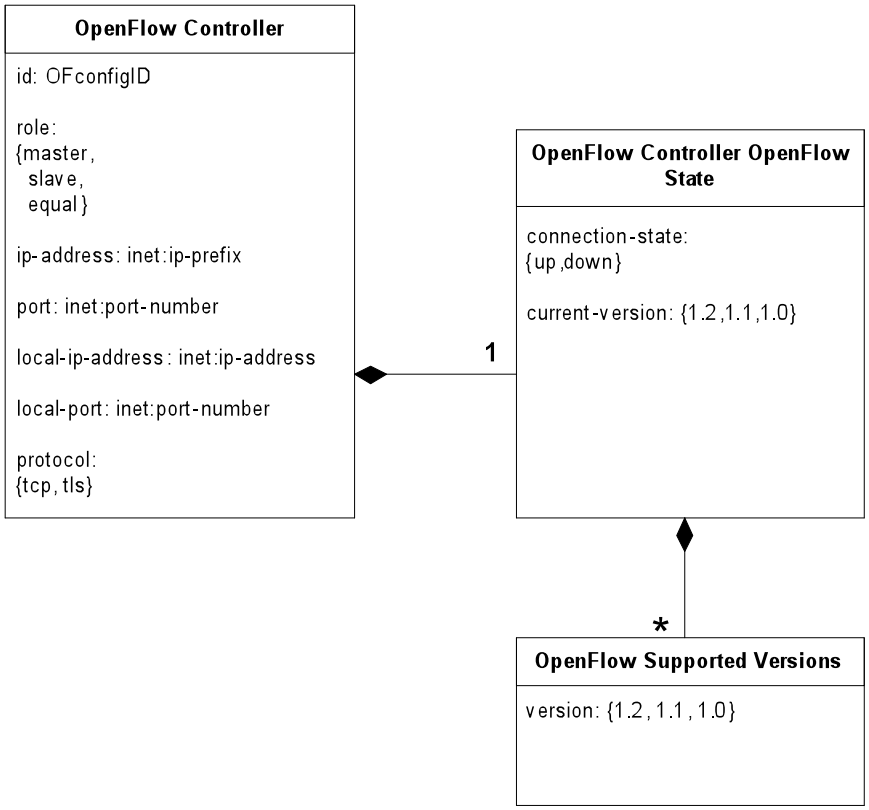

Figure 8: Data Model Diagram for an OpenFlow Controller

7.5.2 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>
```

7.5.3 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>
```

7.5.4 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>.

7.5.5 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.";
      }
    }
}
```

7.6 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.

7.6.1 UML Diagram

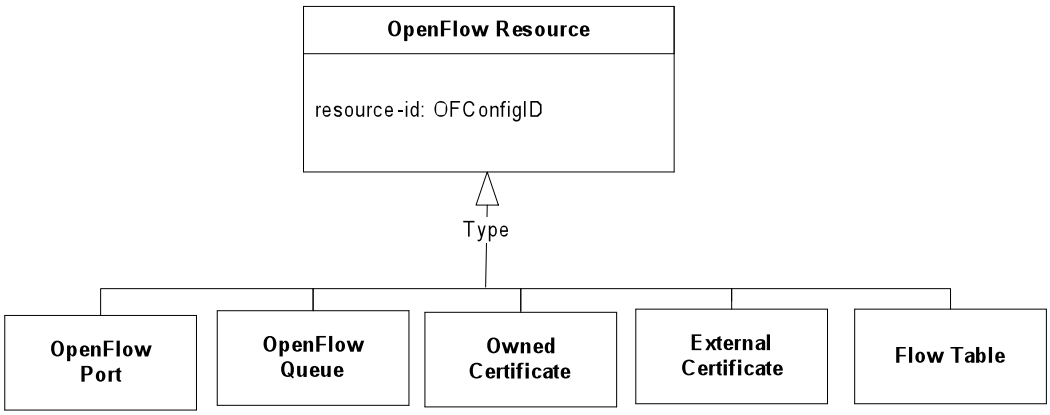

Figure 9: Data Model Diagram for an OpenFlow Resource

7.6.2 XML Schema

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

7.6.3 XML Example

The superclass is not instantiated.

7.6.4 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.

7.6.5 YANG Specification

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

7.7 OpenFlow Port

The OpenFlow Port is an instance of an OpenFlow resource. It may represent a physical port or a logical port. A logical port represents a tunel endpoint as described in the OpenFlow protocol specification.

An OpenFlow Port contains a port configuration object and a port state object. A physical port contains 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. In the case where a port represents a tunnel endpoint, then the port does not contain Port Feature objects, but a Port tunnel object.

7.7.1 UML Diagram

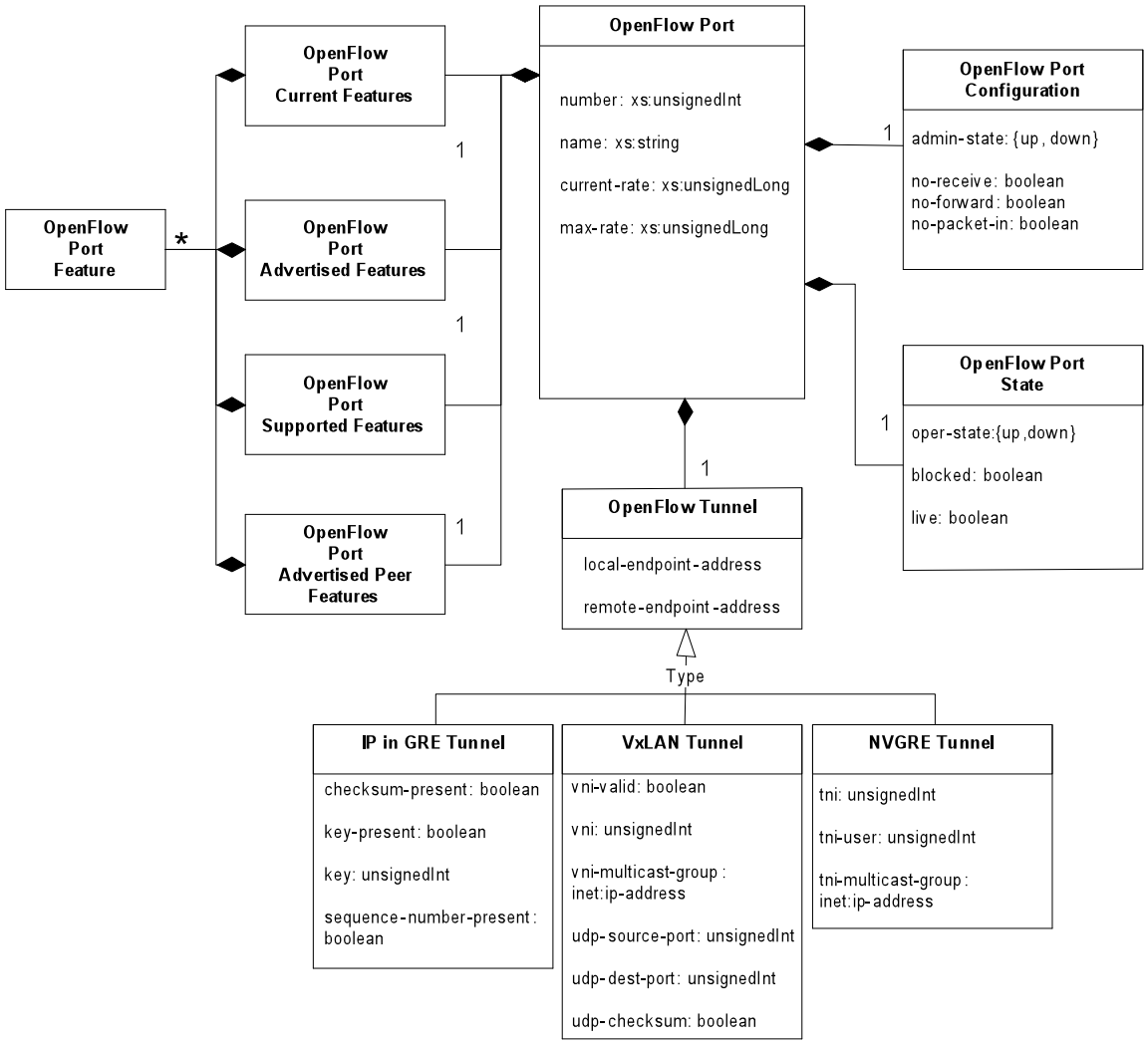

Figure 10: Data Model Diagram for an OpenFlow Port

7.7.2 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"
                minOccurs=”0” maxOccurs=”unbounded”/>
        <xs:choice minOccurs=”0” maxOccurs=”1”>
          <xs:element name=”tunnel”
                type="OFTunnelType"/>
          <xs:element name=”ipgre-tunnel”
                type="OFIPinGREtunnelType"/>
          <xs:element name=”vxlan-tunnel”
                type="OFVxLANTunnelType"/>
          <xs:element name=”nvgre-tunnel”
                type="OFNVGRETunnelType"/>
        </xs:choice>
      </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>

<xs:complexType name="OFTunnelType">
  <xs:sequence>
    <xs:choice>
      <xs:element name=”local-endpoint-ipv4-address”
                type="inet:ipv4-address"/>
```



```
      <xs:element name=”local-endpoint-ipv6-address”
                type="inet:ipv6-address"/>
      <xs:element name=”local-endpoint-mac-address”
                type="inet:mac-address"/>
    </xs:choice>
    <xs:choice>
      <xs:element name=”remote-endpoint-ipv4-address”
                type="inet:ipv4-address"/>
      <xs:element name=”remote-endpoint-ipv6-address”
                type="inet:ipv6-address"/>
      <xs:element name=”remote-endpoint-mac-address”
                type="inet:mac-address"/>
    </xs:choice>
  </xs:sequence>
</xs:complexType>

<xs:complexType name="OFIPinGREtunnelType">
  <xs:complexContent>
    <xs:extension base="OFTunnelType">
      <xs:sequence>
        <xs:element name="checksum-present" type="xs:boolean"
                default="true"/>
        <xs:element name="key-present" type="xs:boolean"
                default="true"/>
        <xs:element name="key" type="xs:unsignedInt"/>
        <xs:element name="sequence-number-present" type="xs:boolean"
                default="false"/>
      </xs:sequence>
    </xs:extension>
  </xs:complexContent>
</xs:complexType>

<xs:complexType name="OFVxLANTunnelType">
  <xs:complexContent>
    <xs:extension base="OFTunnelType">
      <xs:sequence>
        <xs:element name="vni-valid" type="xs:boolean"
                default="true"/>
        <xs:element name="vni" type="xs:unsignedInt"/>
        <xs:element name="vni-multicast-group" type="inet:ip-address"/>
        <xs:element name="udp-source-port" type="xs:unsignedInt"/>
        <xs:element name="udp-dest-port" type="xs:unsignedInt"
                default=IANA_VXLAN_PORT/>
        <xs:element name="udp-checksum" type="xs:boolean"
                default="false"/>
      </xs:sequence>
    </xs:extension>
  </xs:complexContent>
</xs:complexType>

<xs:complexType name="OFNVGRETunnelType">
  <xs:complexContent>
    <xs:extension base="OFTunnelType">
      <xs:sequence>
      <xs:element name="tni" type="xs:unsignedInt"/>
      <xs:element name="tni-user" type="xs:unsignedInt"/>
      <xs:element name="tni-multicast-group"
```



```
                type="inet:ip-address"/>
      </xs:sequence>
    </xs:extension>
  </xs:complexContent>
</xs:complexType>
```

7.7.3 XML Examples

```
<!-- Example for a physical port -->
<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>

<!-- Example for a logical port representing a VxLAN tunnel -->
<port>
  <resource-id>LogicalPort14</resource-id>
  <number>14</number>
  <name>logicalPort14VxLAN</name>
  <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>true</live>
    </state>
    <vxlan-tunnel>
      <local-endpoint-ipv4-address>
        192.0.2.9
      </local-endpoint-ipv4-address>
      <remote-endpoint-ipv4-address>
        192.0.2.112
      </remote-endpoint-ipv4-address>
      <vni-valid>true</vni-valid>
      <vni>15581985</vni>
      <udp-source-port>3804</udp-source-port>
      <udp-dest-port>3801</udp-dest-port>
      <udp-checksum>false</udp-checksum>
    </vxlan-tunnel>
</port>

<!-- Example for a logical port representing a NVGRE tunnel -->
<port>
    <resource-id>LogicalPort17</resource-id>
    <number>17</number>
    <name>logicalPort17NVGRE</name>
    <max-rate>1000</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>true</live>
    </state>
    <nvgre-tunnel>
      <local-endpoint-ipv4-address>
        192.0.2.7
      </local-endpoint-ipv4-address>
      <remote-endpoint-ipv4-address>
        192.0.2.97
      </remote-endpoint-ipv4-address>
      <tni>15581985</tni>
      <tni-resv>173</tni-resv>
    </nvgre-tunnel>
</port>
```

7.7.4 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”.

Port Configuration

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.

Port State

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.

Port Features

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>.

Tunnel (Logical Port)

A tunnel endpoint corresponds to a logical OpenFlow port that supports a specific encapsulation method. A common use case for tunnels is to create virtual overlay networks by encapsulating, for example, Layer 2 (Ethernet) traffic in Layer 3 (IP) packets. OF-CONFIG enables the association of logical OpenFlow ports with an associated tunnel type and corresponding parameters for the tunnel.

Element <tunnel> is only present if the port is a logical port that represents a tunnel endpoint. It contains a tunnel type specific element. Currently defined are the following tunnel types: IPinGRE, VxLAN, and NVGRE. All tunnel types have a common set of contained elements: a local and a remote endpoint address (<local-XXX-address> and <remote-XXX-address>) for address types IPv4, IPv6, and MAC..

IPinGRE Tunnel

For IP-in-GRE tunnels, further elements may be used. The presence of the checksum, key, and sequence number is indicated by boolean elements <checksum-present>, <key-present>, and <sequence-number-present>. Element <key> indicates the key value used. It should not be present if the value of element <key-present> is ”false”. In an implementation of IP-in-GRE tunnels, the <key> element could be used to set the OXM_OF_TUNNEL_ID match field metadata in the OpenFlow protocol.

VXLAN Tunnel

VxLAN tunnel elements are based on the specification current at the time of this writing (draft-mahalingam-dutt-dcops-vxlan-01.txt). The <vni-valid> boolean element indicates how the corresponding flag should be set in packets sent on the tunnel. It SHOULD generally be set to “true”. The <vni> element is the virtual network identifier assigned to all packets sent on the tunnel. ”. A VxLAN implementation may use the <vni> element to set the OXM_OF_TUNNEL_ID match field metadata in the OpenFlow protocol. If IP multicast is used to support broadcast on the tunnel the <vni-multicast-group> element MAY be used to specify the corresponding multicast IP address. The <udp-source-port> element MAY be used to set the outer UDP source port number, e.g., to ensure consistent hashing for ECMP. If <udp-source-port> is absent, it is expected that the source port will be set dynamically during transmission. The <udp-dest-port> SHOULD be set to the IANA assigned well-known port number for VxLAN (pending assignment as of this writing). The <udp-checksum> element is a boolean flag to indicate whether or not the outer UDP checksum should be set. Typically, this element SHOULD be set to “false”.

NVGRE Tunnel

NVGRE tunnel elements are based on the specification current at the time of this writing (draft-sridharan-virtualization-nvgre-00.txt). The <tni> element is the tenant network identifier assigned to all packets sent on the tunnel. NVGRE implementations may map the <tni> element to the OXM_OF_TUNNEL_ID match field metadata in the OpenFlow protocol. The <tni-user> element MAY be present – it is used to set the reserved user-defined bits of the GRE key field, e.g., to introduce entropy for the purposes of exploiting path diversity. If IP multicast is used to support broadcast on the tunnel the <tni-multicast-group> element MAY be used to specify the corresponding multicast IP address.

7.7.5 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.";
    }
  }
  grouping openflow-port-base-tunnel-grouping {
    description "A grouping with information included in every
                supported tunnel type. ";
    choice local-endpoint-address {
      leaf local-endpoint-ipv4-adress {
        type inet:ipv4-address;
        description "The IPv4 address of the local tunnel endpoint.";
      }
      leaf local-endpoint-ipv6-adress {
        type inet:ipv6-address;
        description "The IPv6 address of the local tunnel endpoint.";
      }
      leaf local-endpoint-mac-adress {
        type yang:mac-address;
        description "The MAC address of the local tunnel endpoint.";
      }
    }
    choice remote-endpoint-address {
      leaf remote-endpoint-ipv4-adress {
        type inet:ipv4-address;
        description "The IPv4 address of the remote tunnel endpoint.";
      }
      leaf remote-endpoint-ipv6-adress {
        type inet:ipv6-address;
        description "The IPv6 address of the remote tunnel endpoint.";
      }
      leaf remote-endpoint-mac-adress {
        type yang:mac-address;
        description "The MAC address of the remote tunnel endpoint.";
      }
    }
  }
  choice tunnel-type {
    container tunnel {
      description "Features of a basic IP-in-GRE tunnel.
                Tunnels are modeld as logical ports.";
      uses openflow-port-base-tunnel-grouping;
    }
    container ipgre-tunnel {
      description "Features of a IP-in-GRE tunnel with key,
                checksum, and sequence number information.";
      uses openflow-port-base-tunnel-grouping;
```



```
      leaf checksum-present {
        type boolean;
        description "Indicates presence of the GRE checksum.";
        default true;
      }
      leaf key-present {
        type boolean;
        description "Indicates presence of the GRE key.";
        default true;
      }
      leaf key {
        type uint32;
        description "The (optional) key of the GRE tunnel.";
      }
      leaf sequence-number-present {
        type boolean;
        description "Indicates presence of the GRE sequence number.";
        default false;
      }
    }
    container vxlan-tunnel {
      description "Features of a VxLAN tunnel.";
      uses openflow-port-base-tunnel-grouping;
      leaf vni-valid {
        type boolean;
        description "Indicates how the corresponding flag should be set
          in packets sent on the tunnel";
        default true;
      }
      leaf vni {
        type uint32;
        description "Virtual network identifier assigned to all packets
          sent on the tunnel";
      }
      leaf vni-multicast-group {
        type inet:ip-address;
        description "If IP multicast is used to support broadcast on
          the tunnel this specifies the corresponding multicast IP
          address";
      }
      leaf udp-source-port {
        type inet:port-number;
        description "Specifies the outer UDP source port number .";
      }
      leaf udp-dest-port {
        type inet:port-number;
        description "Specifies the outer UDP destination port number,
          generally the well-known port number for VxLAN";
      }
      leaf udp-checksum {
        type boolean;
        description "Boolean flag to indicate whether or not the outer
          UDP checksum should be set";
        default false;
      }
    }
    container nvgre-tunnel {
```



```
      description "Features of a NVGRE tunnel.";
      uses openflow-port-base-tunnel-grouping;
      leaf tni {
        type uint32;
        description "Specifies the tenant network identifier assigned
          to all packets sent on the tunnel";
      }
      leaf tni-user {
        type uint32;
        description "Used to set the reserved user-defined bits of the
          GRE key field";
      }
      leaf tni-multicast-group {
        type inet:ip-address;
        description "If IP multicast is used to support broadcast on
          the tunnel this specifies the corresponding multicast IP
          address";
      }
    }
  }
}
```

7.8 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.

7.8.1 UML Diagram

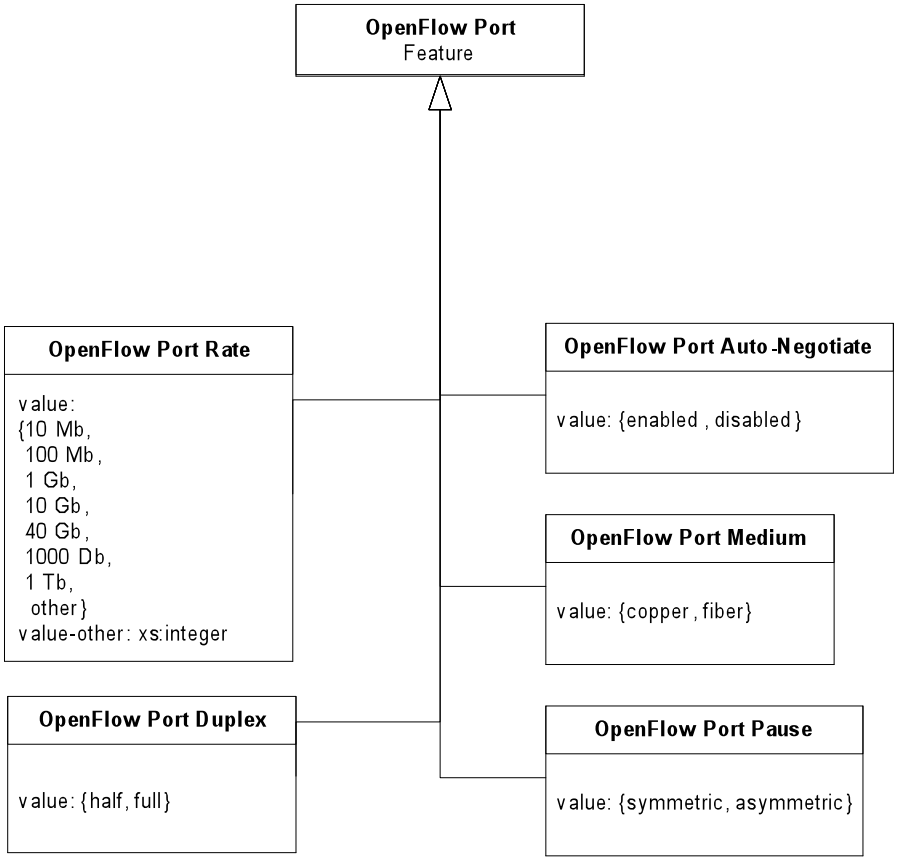

Figure 71: Data Model Diagram for an OpenFlow Port Feature

7.8.2 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>
```

7.8.3 XML Example

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

7.8.4 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>`.

7.8.5 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.";
  }
}
```

7.9 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.

7.9.1 UML Diagram

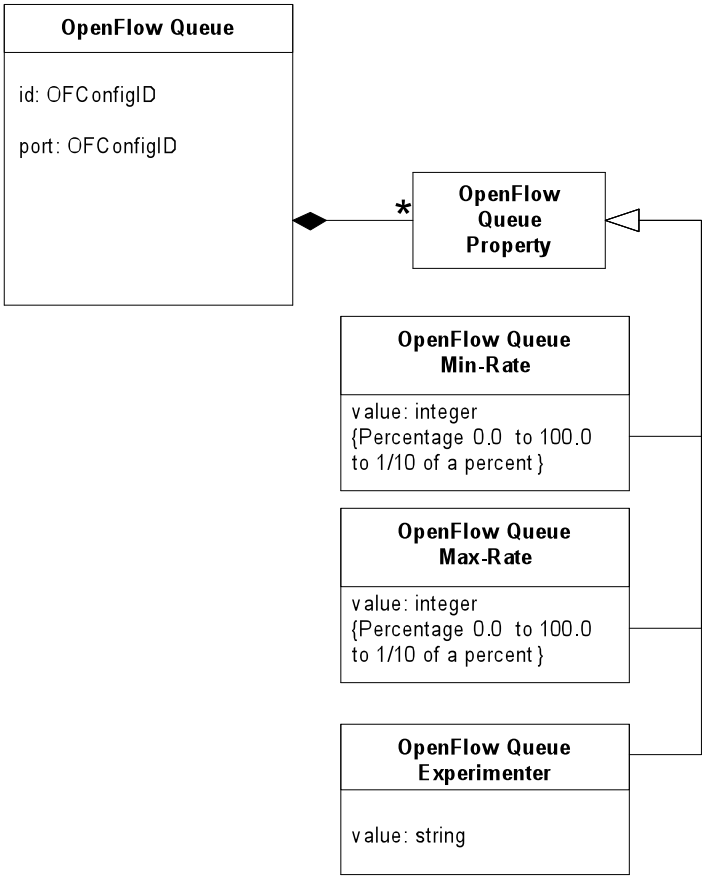

Figure 12: Data Model Diagram for an OpenFlow Queue

7.9.2 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>
```

7.9.3 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>
```

7.9.4 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>.

7.9.5 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.";
    }
  }
}
```

7.10 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.

7.10.1 UML Diagram

Figure 83: Data Model Diagram for a Certificate

7.10.2 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>
```

7.10.3 XML Example

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

7.10.4 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.

7.10.5 YANG Specification

```
grouping openflow-external-certificate-grouping {
  description "This grouping specifies a certificate that can be used
               by an OpenFlow Logical Switch for authenticating a
               controller when a TLS connection is established.";
  leaf resource-id {
    type inet:uri;
    description "A unique but locally arbitrary identifier that
                 identifies an external certificate and is persistent
                 across reboots of the system.";
  }
  leaf certificate {
    type string;
    mandatory true;
    description "An X.509 certificate in DER format base64
                 encoded.";
  }
}
```

7.11 Owned Certificate

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

7.11.1 UML Diagram

Figure 14: Data Model Diagram for Owned Certificate

7.11.2 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>
```

7.11.3 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>
```

7.11.4 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. The private key is encoded as specified in XML-Signature Syntax and Processing (http://www.w3.org/TR/2001/PR-xmldsig-core-20010820/). Currently the specification only support DSA and RSA keys.

7.11.5 YANG Specification

```
grouping openflow-owned-certificate-grouping {
  description "This grouping specifies a certificate and a private key.
               It can be used by an OpenFlow Logical Switch for
               authenticating itself to a controller when a TLS
               connection is established.";
  leaf resource-id {
    type inet:uri;
    description "A unique but locally arbitrary identifier that
                 identifies an external certificate and is persistent
                 across reboots of the system.";
```

```
    }
    leaf certificate {
      type string;
      mandatory true;
      description "An X.509 certificate in DER format base64 encoded.";
    }
    container private-key {
      uses KeyValueType;
      description "tbd.";
    }
}

grouping KeyValueType {
    choice key-type {
      mandatory true;
      case dsa {
        container DSAKeyValue {
          uses DSAKeyValueType;
        }
      }
      case rsa {
        container RSAKeyValue {
          uses RSAKeyValueType;
        }
      }
    }
}

grouping DSAKeyValueType {
    leaf P {
      when "count(../Q) != 0";
      type binary;
      mandatory true;
    }
    leaf Q {
      when "count(../P) != 0";
      type binary;
      mandatory true;
    }
    leaf J {
      type binary;
      mandatory true;
    }
    leaf G {
      type binary;
      mandatory true;
    }
    leaf Y {
      type binary;
      mandatory true;
    }
    leaf Seed {
      when "count(../PgenCounter) != 0";
      type binary;
      mandatory true;
    }
    leaf PgenCounter {
```



```
    when "count(../Seed) != 0";
    type binary;
    mandatory true;
  }
}

grouping RSAKeyValueType {
  leaf Modulus {
    type binary;
    mandatory true;
  }
  leaf Exponent {
    type binary;
    mandatory true;
  }
}
```

7.12 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.

7.12.1 UML Diagram

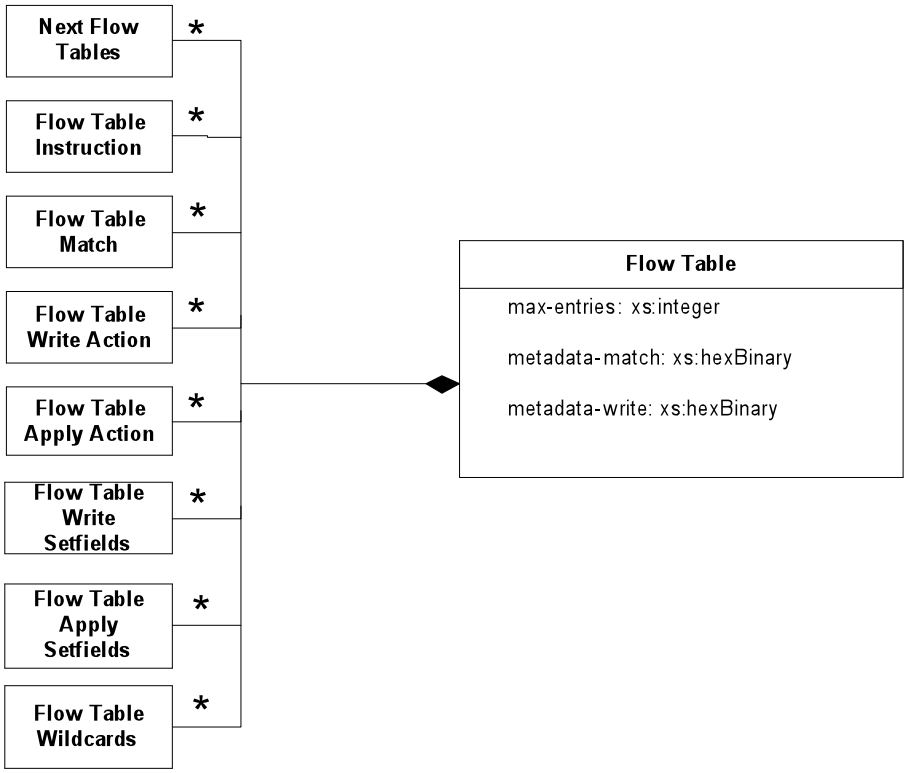

Figure 15: Data Model Diagram for Flow Table

7.12.2 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="next-tables" type="OFNextFlowTables"/>
         <xs:element name="instructions"
           type="OFFlowTableInstructions"/>
         <xs:element name="matches" type="OFFlowTableMatchFields"/>
         <xs:element name="write-actions"
           type="OFFlowTableWriteActions"/>
         <xs:element name="apply-actions"
           type="OFFlowTableApplyActions"/>
         <xs:element name="write-setfields"
           type="OFFlowTableMatchFields"/>
         <xs:element name="apply-setfields"
           type="OFFlowTableMatchFields"/>
         <xs:element name="wildcards" type="OFFlowTableMatchFields"/>
         <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="OFNextFlowTables">
   <xs:sequence minOccurs="1" maxOccurs="unbounded">
     <xs:element name="table-id" type="OFConfigID"/>
   </xs:sequence>
</xs:complexType>

<xs:complexType name="OFFlowTableMatcheFields">
   <xs:sequence minOccurs="1" maxOccurs="unbounded">
     <xs:element name="type" type="OFMatchFieldType"/>
   </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="OFMatchFieldType">
   <xs:annotation>
     <xs:documentation> The open flow match field 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>
```

7.12.3 XML Example

```
<flow-table>
  <resource-id>flowtable1</resource-id>
  <max-entries>255</max-entries>
  <next-tables>
    <table-id>100</table-id>
    <table-id>101</table-id>
  </next-tables>
  <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>
    <write-setfields>
      <type>ethernet-dest</type>
    </write-setfields>
    <apply-setfields>
      <type>ethernet-dest</type>
    </apply-setfields>
    <wildcards>
      <type> udp-dest</type>
    </wildcards>
    <metadata-match>30</metadata-match>
</flow-table>
```

7.12.4 Normative Constraints

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

Element <max-entries> denotes the maximum of flow entries the flow table can support. Due to limitations imposed by modern hardware, the max-entries value should be considered advisory and best effort approximation of the capacity of the table.

Element <next-tables> indicates the array of tables that can be directly reached from the present table using "goto-table" instruction.

Element <instructions> denotes the types of flow instructions supported by the flow table. Flow instructions associated with a flow table entry are executed when a flow matches the flow entry in the flow table.

Element <matches> denotes the types of match fields supported by the flow table. These match fields are defined in OpenFlow Specification version 1.2[1]. An OpenFlow Logical Switch is not required to support all match field types and supported match field types don’t need to be implemented in the same table lookup.

Element <write-actions> specifies the action types which could be merged into the current action set of flow entries of the flow table. The merging operation is performed by “write-action” flow instruction.

Element <apply-actions> specifies the action types which could be immediatedly applied without any change to the action set of flow entries of the flow table. The applying operation is performed by “apply-action” flow instruction.

Element <write-setfields> specifies "set-field" action types supported by the table using "write-actions" instruction.

Element <apply-setfields> specifies "set-field" action types supported by the table using "apply-actions" instruction.

Element <wildcards> specifies the fields for which the table supports wildcarding(omitting).

Element <metadata-match> indicates the bits of the metadata field that the table can match on. It is represented as 64-bit integer in hexadecimal digits([0-9a-fA-F]) format.

Element <metadata-write> indicates the bits of the metadata field that the table can write using the “write-metadata” instruction. It is represented as 64-bit integer in hexadecimal digits([0-9a-fA-F]) format.

7.12.5 YANG Specification

```
typedef match-field-type {
  description "The types of match fields 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;
    num 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 next-tables {
    leaf-list table-id {
      type inet:uri;
    }
    description "The array of flow table ids that can be directly
      reached from the present table using "goto-table"
      instruction.";
  }
  container instructions {
    leaf-list type {
      type instruction-type;
    }
    description "The instruction types supported by the flow table."
  }
  container matches {
    leaf-list type {
      type match-field-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."
  }
  container write-setfields {
    leaf-list type {
      type match-field-type;
    }
    description "'set-field' action types supported by the table
      using 'write-actions' instruction.";
  }
  container apply-setfields {
    leaf-list type {
      type match-field-type;
    }
    description "'set-field' action types supported by the table
      using 'apply-actions' instruction.";
  }
  container wildcards {
    leaf-list type {
      type match-field-type;
    }
    description "The fields for which the table supports
      wildcarding(omitting).";
  }
  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."
  }
}
```

8 Binding to NETCONF

The OF-CONFIG1.1 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-CONFIG1.1 requires that devices supporting OF-CONFIG1.1 MUST implement NETCONF protocol (4) as the transport. This in turn implies as specified by NETCONF specification that OpenFlow Capable Switches supporting OF-CONFIG1.1 must implement SSH as a transport protocol. In addition, the OpenFlow Capable Switches implementing OF-CONFIG1.1 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-CONFIG1.1. NETCONF defines a set of operations on top of a messaging layer (RPC). Below diagram shows the various layers of NETCONF protocol.

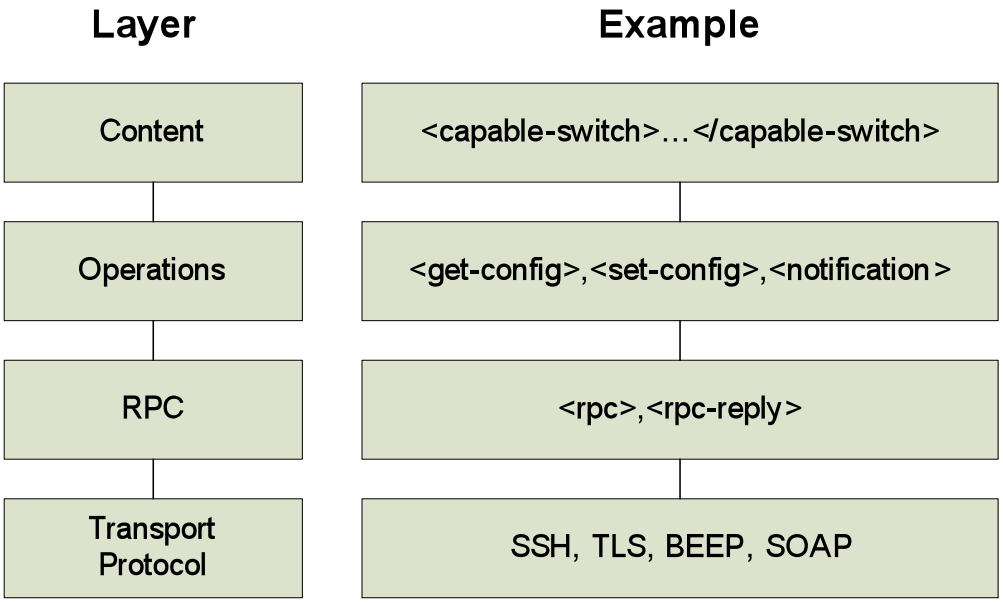

Figure 96 NETCONF Layers and Examples

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 1.1 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.

8.1 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

8.1.1 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.

8.1.2 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>
```

8.1.3 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>
```

8.1.4 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>
```

8.2 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>
```


Appendix A 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:element name="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="next-tables" type="OFNextFlowTables"/>
        <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="write-setfields" type="OFFlowTableMatchFields"/>
        <xs:element name="apply-setfields" type="OFFlowTableMatchFields"/>
        <xs:element name="wildcards" type="OFFlowTableMatchFields"/>
        <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="OFNextFlowTables">
  <xs:sequence minOccurs="1" maxOccurs="unbounded">
    <xs:element name="table-id" type="OFConfigID"/>
  </xs:sequence>
</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="OFFlowTableMatchFields">
  <xs:sequence minOccurs="1" maxOccurs="unbounded">
    <xs:element name="type" type="OFMatchFieldType"/>
  </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="OFMatchFieldType">
  <xs:annotation>
    <xs:documentation> The open flow match field 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>
```

Appendix B YANG Specification

```
module onf-config1.1 {
  namespace "urn:onf:of11:config:yang";
  prefix of11-config;

  import ietf-yang-types { prefix yang; }
  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 {
    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;
    }
    description "The types of actions defined in OpenFlow Switch
      Specification version 1.2.";
  }

  typedef instruction-type {
    type enumeration {
      enum apply-actions;
      enum clear-actions;
      enum write-actions;
      enum write-metadata;
      enum goto-table;
    }
    description "The types of instructions defined in OpenFlow Switch
      Specification version 1.2.";
  }

  typedef match-field-type {
    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;
    }
    description "The types of matches defined in OpenFlow Switch
      Specification version 1.2.";
  }
  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 certificate {
        type leafref {
          path "/capable-switch/resources/owned-certificate/resource-
            id";
        }
        description "The reference to the owned certificate in the
          OpenFlow Capable Switch that the OpenFlow Logical Switch used
          to identify itself.";
      }
      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-port-base-tunnel-grouping {
      description "A grouping with information included in every
        supported tunnel type.";
      choice local-endpoint-address {
        leaf local-endpoint-ipv4-adress {
          type inet:ipv4-address;
          description "The IPv4 address of the local tunnel endpoint.";
        }
        leaf local-endpoint-ipv6-adress {
          type inet:ipv6-address;
          description "The IPv6 address of the local tunnel endpoint.";
        }
        leaf local-endpoint-mac-adress {
          type yang:mac-address;
          description "The MAC address of the local tunnel endpoint.";
        }
      }
      choice remote-endpoint-address {
        leaf remote-endpoint-ipv4-adress {
          type inet:ipv4-address;
          description "The IPv4 address of the remote tunnel endpoint.";
        }
        leaf remote-endpoint-ipv6-adress {
          type inet:ipv6-address;
          description "The IPv6 address of the remote tunnel endpoint.";
        }
        leaf remote-endpoint-mac-adress {
          type yang:mac-address;
          description "The MAC address of the remote tunnel endpoint.";
        }
      }
    }
    choice tunnel-type {
      container tunnel {
        description "Features of a basic IP-in-GRE tunnel. Tunnels are
          modeld as logical ports.";
        uses openflow-port-base-tunnel-grouping;
      }
      container ipgre-tunnel {
        description "Features of a IP-in-GRE tunnel with key, checksum,
          and sequence number information.";
        uses openflow-port-base-tunnel-grouping;
        leaf checksum-present {
          type boolean;
          description "Indicates presence of the GRE checksum.";
          default true;
        }
        leaf key-present {
          type boolean;
          description "Indicates presence of the GRE key.";
          default true;
        }
        leaf key {
          type uint32;
          description "The (optional) key of the GRE tunnel.";
```



```
        }
        leaf sequence-number-present {
          type boolean;
          description "Indicates presence of the GRE sequence number.";
          default false;
        }
      }
      container vxlan-tunnel {
        description "Features of a VxLAN tunnel.";
        uses openflow-port-base-tunnel-grouping;
        leaf vni-valid {
          type boolean;
          description "Indicates how the corresponding flag should be
            set in packets sent on the tunnel";
          default true;
        }
        leaf vni {
          type uint32;
          description "Virtual network identifier assigned to all
            packets sent on the tunnel";
        }
        leaf vni-multicast-group {
          type inet:ip-address;
          description "If IP multicast is used to support broadcast on
            the tunnel this specifies the corresponding multicast IP
            address";
        }
        leaf udp-source-port {
          type inet:port-number;
          description "Specifies the outer UDP source port number .";
        }
        leaf udp-dest-port {
          type inet:port-number;
          description "Specifies the outer UDP destination port number,
            generally the well-known port number for VxLAN";
        }
        leaf udp-checksum {
          type boolean;
          description "Boolean flag to indicate whether or not the outer
            UDP checksum should be set";
          default false;
        }
      }
      container nvgre-tunnel {
        description "Features of a NVGRE tunnel.";
        uses openflow-port-base-tunnel-grouping;
        leaf tni {
          type uint32;
          description "Specifies the tenant network identifier assigned
            to all packets sent on the tunnel";
        }
        leaf tni-user {
          type uint32;
          description "Used to set the reserved user-defined bits of the
            GRE key field";
        }
        leaf tni-multicast-group {
```



```
          type inet:ip-address;
          description "If IP multicast is used to support broadcast on
            the tunnel this specifies the corresponding multicast IP
            address";
        }
      }
    }
  }

  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-external-certificate-grouping {
    description "This grouping specifies a certificate that can be
      used by an OpenFlow Logical Switch for authenticating a
      controller when a TLS connection is established.";
    leaf resource-id {
      type inet:uri;
      description "A unique but locally arbitrary identifier that
        identifies an external certificate and is persistent across
        reboots of the system.";
    }
    leaf certificate {
      type string;
      mandatory true;
      description "An X.509 certificate in DER format base64
        encoded.";
    }
  }

  grouping openflow-owned-certificate-grouping {
    description "This grouping specifies a certificate and a private
      key. It can be used by an OpenFlow Logical Switch for
      authenticating itself to a controller when a TLS connection is
      established.";
    leaf resource-id {
      type inet:uri;
      description "A unique but locally arbitrary identifier that
        identifies an external certificate and is persistent across
        reboots of the system.";
    }
    leaf certificate {
      type string;
      mandatory true;
      description "An X.509 certificate in DER format base64
        encoded.";
    }
    container private-key {
      uses KeyValueType;
      description "tbd.";
    }
  }

  grouping KeyValueType {
    choice key-type {
      mandatory true;
      case dsa {
        container DSAKeyValue {
          uses DSAKeyValueType;
        }
      }
```



```
      case rsa {
        container RSAKeyValue {
          uses RSAKeyValueType;
        }
      }
    }
  }

  grouping DSAKeyValueType {
    leaf P {
      when "count(../Q) != 0";
      type binary;
      mandatory true;
    }
    leaf Q {
      when "count(../P) != 0";
      type binary;
      mandatory true;
    }
    leaf J {
      type binary;
      mandatory true;
    }
    leaf G {
      type binary;
      mandatory true;
    }
    leaf Y {
      type binary;
      mandatory true;
    }
    leaf Seed {
      when "count(../PgenCounter) != 0";
      type binary;
      mandatory true;
    }
    leaf PgenCounter {
      when "count(../Seed) != 0";
      type binary;
      mandatory true;
    }
  }

  grouping RSAKeyValueType {
    leaf Modulus {
      type binary;
      mandatory true;
    }
    leaf Exponent {
      type binary;
      mandatory true;
    }
  }

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 next-tables {
      leaf-list table-id {
        type inet:uri;
      }
      description "The array of flow table ids that can be directly
        reached from the present table using 'goto-table'
        instruction.";
    }
    container instructions {
      leaf-list type {
        type instruction-type;
      }
      description "The instruction types supported by the flow
        table.";
    }
    container matches {
      leaf-list type {
        type match-field-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.";
    }
    container write-setfields {
      leaf-list type {
        type match-field-type;
      }
      description "'set-field' action types supported by the table
      using 'write-actions' instruction.";
    }
    container apply-setfields {
      leaf-list type {
        type match-field-type;
      }
      description "'set-field' action types supported by the table
```



```
      using 'apply-actions' instruction.";
    }
    container wildcards {
      leaf-list type {
        type match-field-type;
      }
      description " the fields for which the table supports
      wildcarding(omitting).";
    }
    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 owned-certificate {
        key "resource-id";
        unique "resource-id";
        description "The list contains all owned certificate resources
          of the OpenFlow Capable Switch.";
        uses openflow-owned-certificate-grouping;
      }
      list external-certificate {
        key "resource-id";
        unique "resource-id";
        description "The list contains all external certificate
          resources of the OpenFlow Capable Switch.";
        uses openflow-external-certificate-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;
      }
    }
  }
}
```

Appendix C Bibliography

1. *OpenFlow Specification 1.3.* **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.

Appendix D 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.
rev10	3/21/12	Chuan	Updated UML diagrams. Added new diagrams for certificate, capabilities and flow table
rev15	3/26/12	Chuan	Update the Normative constraint changes I added in 7.3.4, 7.4.4, 7.12.4
rev16	3/26/12	Carl	Formatted new material.

Appendix E 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	