|  |
| --- |
|  |
| OF-CONFIG 1.2 (DRAFT) |
| OpenFlow Management and Configuration Protocol |
|  |
|  |
|  |

Contact: Deepak Bansal ([dbansal@microsoft.com](mailto:dbansal@microsoft.com)), Stuart Bailey ([sbailey@infoblox.com](mailto:sbailey@infoblox.com)), Thomas Dietz ([Thomas.Dietz@neclab.eu](mailto:Thomas.Dietz@neclab.eu)), Anees Shaikh (aashaikh@us.ibm.com) , Carl Moberg (calle@tail-f.com)

Copyright © 2013 Open Networking Foundation

Disclaimer

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

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

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

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

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

Contents

1 Introduction 5

2 Motivation 5

2.1 OF-CONFIG and OF-SWITCH 7

3 Scope 8

4 Normative Language 9

5 Terms 9

5.1 OpenFlow Capable Switch 9

5.2 OpenFlow Configuration Point 9

5.3 OpenFlow Logical Switch 10

5.4 OpenFlow Resource 10

5.4.1 OpenFlow Queue 10

5.4.2 OpenFlow Port 10

5.5 OpenFlow Controller 10

5.6 NDM 10

6 Requirements 10

6.1 Requirements from the OpenFlow 1.3 Protocol Specification 10

6.1.1 Instantiation of one or more Openflow Data Planes on an Openflow Capable Switch and assignment of resources to these data planes 11

6.1.2 Connection Setup to a Controller 11

6.1.3 Multiple Controllers 11

6.1.4 OpenFlow Logical Switches 11

6.1.5 Connection Interruption 11

6.1.6 Encryption 12

6.1.7 Queues 12

6.1.8 Ports 12

6.1.9 Capability Discovery 13

6.1.10 Datapath ID 13

6.2 Requirements for NDMs 13

6.3 Operational Requirements 14

6.4 Requirements for the Switch Management Protocol 14

7 NETCONF as the Transport Protocol 15

8 Data Model 17

8.1 YANG Module 18

8.2 Core Data Model 18

8.3 OpenFlow Capable Switch 19

8.3.1 UML Diagram 19

8.3.2 XML Example 20

8.4 OpenFlow Configuration Point 20

8.4.1 UML Diagram 20

8.4.2 XML Example 21

8.5 OpenFlow Logical Switch 21

8.5.1 UML Diagram 21

8.5.2 XML Example 22

8.6 Logical Switch Capabilities 23

8.6.1 UML Diagram 23

8.6.2 XML Example 23

8.7 OpenFlow Controller 24

8.7.1 UML Diagram 24

8.7.2 XML Example 25

8.8 OpenFlow Resource 25

8.8.1 UML Diagram 25

8.8.2 XML Example 25

8.9 OpenFlow Port 26

8.9.1 UML Diagram 26

8.9.2 XML Examples 27

8.10 OpenFlow Port Feature 28

8.10.1 UML Diagram 29

8.10.2 XML Example 29

8.11 OpenFlow Queue 29

8.11.1 UML Diagram 30

8.11.2 XML Example 30

8.12 External Certificate 31

8.12.1 UML Diagram 31

8.12.2 XML Example 31

8.13 Owned Certificate 31

8.13.1 UML Diagram 32

8.13.2 XML Example 32

8.14 OpenFlow Flow Table 33

8.14.1 UML Diagram 33

8.14.2 XML Example 33

8.15 NDM 34

8.15.1 UML Diagram 34

8.15.2 XML Schema 34

8.15.3 XML Example 36

9 Binding to NETCONF 36

9.1 Requirements 36

9.2 How the Data Model is Bound to NETCONF 37

9.2.1 edit-config 37

9.2.2 get-config 39

9.2.3 copy-config 40

9.2.4 delete-config 40

9.3 RPC error 41

Appendix A Bibliography 43

Appendix B OF-CONFIG XML Schema 44

# 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 (or previous versions) 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 1.1.1 of OF-CONFIG.



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

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

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

# Motivation

The OpenFlow protocol assumes that an OpenFlow datapath (e.g. an Ethernet switch which supports the OpenFlow protocol) has been configured with various artifacts such as the IP addresses of OpenFlow controllers. The motivation for the OpenFlow Configuration Protocol (OF-CONFIG) is to enable the remote configuration of OpenFlow datapaths. While the OpenFlow protocol generally operates on a time-scale of a flow (i.e. as flows are added and deleted), OF-CONFIG operates on a slower time-scale. An example is building forwarding tables and deciding forwarding actions which are done via Openflow protocol while enabling/disabling a port generally 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 introduces an operating context called an OpenFlow Capable Switch for one or more OpenFlow datapaths. 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 between multiple OpenFlow Logical Switches such that each OpenFlow Logical Switch can assume full control over the resources that is assigned to it.

OF-CONFIG 1.2 makes simplifying assumptions about the architecture of OpenFlow switches. The specification is deliberately decoupled from whether the switch supports flowvisor or other virtualization models, for example.

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 provided by software acting as an OpenFlow controller or it may by a service provided by a traditional network management framework. Interactions between the OpenFlow Configuration Points and OpenFlow controllers is outside the scope of OF-CONFIG 1.2, but is expected to be addressed in future versions of the specification.

Figure 2 shows the basic abstractions detailed in OF-CONFIG 1.2 and the lines indicate that the OpenFlow Configuration Points and OpenFlow Capable Switches communicate via OF-CONFIG. The configuration settings then take effect on targeted logical switch(es). OpenFlow Controllers and OpenFlow Logical Switches (i.e. datapaths) communicate via OpenFlow.



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

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

## OF-CONFIG and OF-SWITCH

Although OF-CONFIG is considered a complementary protocol to the main OpenFlow switch specification (OF-SWITCH), it is useful to describe the differences that motivate the need for a separate protocol specification in ONF. The table below summarizes the key differences.

|  |  |  |
| --- | --- | --- |
|  | OpenFlow | OF-CONFIG |
| Primary purpose | Modification of match-action rules effecting flows of network packets across an OpenFlow datapath | Remote configuration of possibly multiple OpenFlow datapaths on a physical or virtual platform |
| Terminology | In the OpenFlow 1.3.1 specification and earlier, the terms OpenFlow datapath and OpenFlow Switch are essentially interchangeable | In the OF-Config 1.1.1 specification and earlier, the term OpenFlow Logical Switch maps to the OpenFlow datapath in the OpenFlow 1.3.1 and earlier specs. The term OpenFlow Capable Switch introduces a new abstraction.  OpenFlow Logical Switch = OpenFlow datapath (as defined in the OpenFlow specification)  OpenFlow Controller = controller (as defined in the OpenFlow specification)  OpenFlow Capable Switch = a new element  OpenFlow Configuration Point = a new element |
| Transport | A bit-level protocol specified in the OpenFlow standard currently supported over TCP, TLS, or SSL | An XML data model and operational behavior specified in the OF-Config standard bound to the NETCONF operations and transport standard for network device configuration and management |
| Protocol endpoints | 1) An OpenFlow datapath, also referred to as OpenFlow Logical Switch (OFLS)  2) An OpenFlow Controller (OFC) | 1) An OpenFlow Capable Switch (OFCS) able to instantiate one or more OpenFlow Logical Switches (i.e. OpenFlow datapaths)  2) An OpenFlow Configuration Point (OFCP) |
| Example usage | An OpenFlow Controller adds a flow modification to an OpenFlow datapath (OFLS) which allows Ethernet frames containing IP packets which originated from 192.168.3.10 and are coming in on the datapath's port 2 to be forwarded out on the datapath's port 14 | An OpenFlow Configuration Point configures a particular OpenFlow Logical Switch (OF datapath) to be associated with a particular OpenFlow Controller |

It should be noted that some properties of the OpenFlow Logical Switch (i.e. OpenFlow datapath) are available and configurable via both OpenFlow and OF-CONFIG. Having multiple channels for some of the same data is considered appropriate since the primary purpose of OpenFlow Configuration Points and OpenFlow Controllers are quite different. The ONF strives for synchronization of the data models and semantics between the OpenFlow and OF-CONFIG standards prior to update of either standard.

# Scope

OF-CONFIG 1.2 is focused on the following functions needed to configure an OpenFlow 1.3 datapath:

* The assignment of one or more OpenFlow controllers to openflow data planes
* 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 set of specific tunnel types such as IP-in-GRE, NV-GRE, VxLAN

New functionality introduced in OF-CONFIG 1.2 includes:

* Instantiation of openflow data planes (called openflow logical switches)
* Assignment of resources of an OpenFlow Capable Switch to one or more OpenFlow Logical Switches
* Support for the emerging Negotiable Datapath Model (NDM) being developed in the ONF
* Versioning Support

Other functions and/or the description of their use have been improved.

While limited in scope, OF-CONFIG 1.2 lays the foundation on top of which various automated and more advanced configurations will be possible in future revisions. The ONF Configuration and Management working group will publish additional specifications for network operations, administration, and management (OAM), including, topology discovery, event management, and bootstrap of the OpenFlow capable network.

Note that even though this specification refers to OpenFlow 1.3, OF-CONFIG 1.2 supports previous OpenFlow versions, specifically, OpenFlow 1.0, 1.1 and 1.2.

# Normative Language

The key words "MUST", "MUST NOT", "REQUIRED", "SHALL", "SHALL NOT", "SHOULD", "SHOULD NOT", "RECOMMENDED", "MAY", and "OPTIONAL" in this document are to be interpreted as described in RFC 2119 (3).

# Terms

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

## OpenFlow Capable Switch

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

## OpenFlow Configuration Point

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

## OpenFlow Logical Switch

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

## OpenFlow Resource

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

### OpenFlow Queue

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

### OpenFlow Port

An OpenFlow Port is a forwarding interface of an OpenFlow Logical Switch as described in the OpenFlow specification as the port component of an OpenFlow datapath. An Openflow Port may map to a physical port on a physical switch or a logical port on a physical or virtual switch.

## OpenFlow Controller

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

## NDM

A Negotiable Datapath Model (NDM) is an abstract switch model that describes specific switch forwarding behaviors controllable via the OpenFlow-Switch protocol. The NDM describes specific requirements for switch behavior so that implementers can perform optimizations or deliver more complex forwarding behaviors (beyond what can be scalably represented in a single OpenFlow table) than they could otherwise.

# Requirements

This section describes requirements for the design of OF-CONFIG 1.2.

## 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 within (1). OF-CONFIG 1.2 enables other configuration of OpenFlow switches. The specification of OF-CONFIG 1.2 is written with extensibility in mind. This includes versioning and backward compatibility.

### Instantiation of one or more Openflow Data Planes on an Openflow Capable Switch and assignment of resources to these data planes

An openflow capable switch is capable of hosting one or more openflow data planes (also refered to as openflow logical switch). Initially, the openflow capable switch owns all the resources of the switch and does not have any data plane instantiated. Using the of-config 1.2 protocol, OFCP can instantiate one or more openflow data planes and can assign resources such as queues and ports to these openflow data planes. Some of the resources like management port may not be assigned to any openflow data plane.

### Connection Setup to a Controller

Section 6.2 (Connection Setup) of (1) requires that an OpenFlow switch always initiate the connection to the OpenFlow controller and 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 1.2 must provide means for configuring these parameters. Note that in future, alternative mechanisms for discovering the OpenFlow controller may be supported.

### Multiple Controllers

Section 6.3 of (1) discusses how a switch deals with multiple controllers simultaneously. This implicitly requires OF-CONFIG 1.2 to provide means for configuring multiple instances of the parameter set listed in 6.1.1 for specifying the connection setup to multiple controllers.

### OpenFlow Logical Switches

The OpenFlow 1.3 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.

### Connection Interruption

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

* fail secure mode
* fail standalone mode

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

### Encryption

Section 6.5 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.

### 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 1.2 must provide means for configuring these parameters.

### 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 a requirement for OF-CONFIG 1.2. Section A.3.4.3 of (1) defines the following parameters for port configuration:

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

OF-CONFIG 1.2 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 cannot 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 1.2 must provide means for configuring these advertised features and for obtaining current, supported and peer-advertised state information for these 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. These logical ports may be used in for example, datacenter scenarios for setting up virtual networks. OF-CONFIG 1.2 must support the configuration of these logical ports. However, the configuration of logical ports in OF-CONFIG 1.2 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.

### 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 1.2, 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.

### 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.

## Requirements for NDMs

OF-CONFIG 1.2 includes optional support for Negotiable Datapath Models (NDMs). An NDM is an abstract switch model that describes specific switch forwarding behaviors controllable via the OpenFlow-Switch protocol.

When a capable switch implements the NDM framework (which is an optional enhancement to OpenFlow), an OFCP and a capable switch agree on an NDM to be associated with a logical switch prior to sending control messages, such as flowmods, to the logical switch. This agreement may be implicit (i.e., each side is configured a priori) or negotiated when the control relationship is established.

NDMs are characterized by parameters related to table sizes or optional functionality. The NDM framework allows for implementations to have a range of flexibility in their parameters. Some implementations may have no flexibility; others will allow some adjustment of parameters at the time the OFCP associates the NDM with a logical switch. NDM implementations that support parameter adjustment should (?) also offer an RPC mechanism to allow the OFCP and the capable (?) switch to determine the parameters in a specific situation.

The NDM framework simplifies the job of implementing an OpenFlow controller or OpenFlow agent for a switch. The NDM describes specific requirements for switch behavior so that implementers can perform optimizations or deliver more complex forwarding behaviors (beyond what can be scalably represented in a single OpenFlow table) than they could otherwise.

The optional NDM manageability feature must support the following requirements:

1. The ability to query the capable switch about support for NDMs
2. The ability to query the capable switch for the set of available supported NDMs
3. The ability to associate a logical switch with a parameterized NDM
4. The ability to remove a parameterized NDM from a logical switch

## Operational Requirements

The OF-CONFIG 1.2 must meet support the following scenarios:

1. OF-CONFIG 1.2 must support an OpenFlow Capable Switch being configured by multiple OpenFlow Configuration Points.
2. OF-CONFIG 1.2 must support an OpenFlow Configuration Point managing multiple OpenFlow Capable Switches.
3. OF-CONFIG 1.2 must support an OpenFlow Logical Switch being controlled by multiple OpenFlow Controllers.
4. OF-CONFIG 1.2 must support configuring ports and queues of an OpenFlow Capable Switch that have been assigned to an OpenFlow Logical Switch.
5. OF-CONFIG 1.2 must support discovery of capabilities of an OpenFlow Logical Switch.
6. OF-CONFIG 1.2 must support configuration of tunnels such as IP-in-GRE, NVGRE and VxLan that are represented as logical ports of an OpenFlow Logical Switch.

## Requirements for the Switch Management Protocol

OF-CONFIG 1.2 defines a communication standard between an OpenFlow switch and an OpenFlow Configuration Point. It consists of a network management protocol specified in Section 7 and a data model defined in Section 8. 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 1.2, 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.

# NETCONF as the Transport Protocol

The OF-CONFIG1.1.1 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-CONFIG1.1.1 requires that devices supporting OF-CONFIG1.1.1 MUST implement the NETCONF protocol (4) as their transport protocol. This in turn implies as specified by the NETCONF specification that OpenFlow Capable Switches supporting OF-CONFIG1.1.1 must implement SSH as a transport protocol. In addition, the OpenFlow Capable Switches implementing OF-CONFIG1.1.1 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-CONFIG1.1.1. NETCONF defines a set of operations on top of a messaging layer (RPC). The diagram below shows the various layers of the NETCONF protocol.



Figure 6 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 the next versions of this document.

The NETCONF protocol meets the OF-CONFIG 1.2 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 the 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 provide 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 standards 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.

# Data Model

This section specifies the data model for OF-CONFIG 1.2. 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. an example for XML code encoding an instance of the class

The full XML schema is listed in Appendix A . Normative for OF-CONFIG 1.2 is the XML schema in 2013 and the normative constraints in the descriptions of the individual elements.

One of the design goals of the model is efficient and clear encoding 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.

## YANG Module

OF-CONFIG 1.2 has a companion , distributed as a separate file to aid in implementation of the OF-CONFIG data model. It The YANG module conforms to the normative constraints given in XML schema of 2013 and the additional explanations in this section. Most of the constraints that are given in the description of the XML schema are automatically enforced in the YANG module by syntax elements already built into the YANG language. Implementers that already use the NETCONF tools could profit by using the YANG module to reduce implementation time. Nevertheless, they need to ensure that all normative constraints are obeyed – including those that are not expressible by the YANG syntax.

## Core Data Model

The following UML diagram describes the top-level classes of the data model.



Figure : UML Class Diagram for OF-CONFIG Data Model

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

The switch contains a set of resources of different types. For OF-CONFIG 1.2, 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 1.2.

## OpenFlow Capable Switch

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

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

### UML Diagram



Figure : Data Model Diagram for OpenFlow Capable Switch



### XML Example

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

## 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.

### UML Diagram



Figure : Data Model Diagram for an OpenFlow Configuration Point



### XML Example

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

## OpenFlow Logical Switch

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

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

### UML Diagram



Figure : Data Model Diagram for an OpenFlow Logical Switch

### 



### XML 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> |

## Logical Switch Capabilities

### UML Diagram





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

### 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> |

## OpenFlow Controller

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

### UML Diagram



Figure 8: Data Model Diagram for an OpenFlow Controller



### 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> |

## OpenFlow Resource

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

### UML Diagram



Figure 9: Data Model Diagram for an OpenFlow Resource



### XML Example

The superclass is not instantiated.

## 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.

### UML Diagram



Figure 10: Data Model Diagram for an OpenFlow Port



### 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>4789</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>  <vsid>15581985</vsid>  <flow-id>335</flow-id>  </nvgre-tunnel>  </port> |

## OpenFlow Port Feature

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

### UML Diagram





Figure 1: Data Model Diagram for an OpenFlow Port Feature

### XML Example

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

## OpenFlow Queue

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

### UML Diagram





Figure 12: Data Model Diagram for an OpenFlow Queue

### 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> |

## External Certificate

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

### UML Diagram





Figure 3: Data Model Diagram for a Certificate

### XML Example

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

## 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.

### UML Diagram





Figure 14: Data Model Diagram for Owned Certificate

### 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> |

## OpenFlow Flow Table

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

### UML Diagram





Figure 15: Data Model Diagram for Flow Table

### 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> |

## NDM

A Negotiable Datapath Model (NDM) is an abstract switch model that describes specific switch forwarding behaviors controllable via the OpenFlow-Switch protocol. When using the NDM framework (an optional enhancement to OpenFlow), an OFCP and a capable switch agree on an NDM to be associated with a logical switch prior to sending control messages, such as flowmods, to the logical switch. (need example)

### UML Diagram

### XML Schema

This XML Schema represents the basic placeholder structure for NDMs. Implementations are expected to extend this schema with a set of NDM-specific data definitions (e.g. L2+L3). The details of the specific NDMs are outside the scope of this document. (add reference to NDM paper)

|  |
| --- |
| <xs:group name="a0">  <xs:sequence>  <xs:element name="ndm">  <xs:annotation>  <xs:documentation>  The container for the capable-switch global list of available  NDMs.  </xs:documentation>  </xs:annotation>  <xs:complexType>  <xs:sequence>  <xs:element name="available-ndms" minOccurs="0"  maxOccurs="unbounded">  <xs:annotation>  <xs:documentation>  The list of all NDMs implemented by the capable-switch  Clients can query this list to learn about supported  NDMs before implementing them on logical-switches  </xs:documentation>  </xs:annotation>  <xs:complexType>  <xs:sequence>  <xs:element name="name" type="xs:string">  <xs:annotation>  <xs:documentation>  A unique name of the NDM.  This name corresponds to the  'ndm-implementation-choice' name per  logical-switch. Clients are expected to:  - Find out about available NDMs by querying this  list;  - and configure specific NDM implementations per  logical-switch using this name  </xs:documentation>  </xs:annotation>  </xs:element>  <xs:element name="type">  <xs:annotation>  <xs:documentation>  The NDM type.  </xs:documentation>  </xs:annotation>  <xs:simpleType>  <xs:restriction base="xs:string">  <xs:enumeration value="ttp"/>  <xs:enumeration value="fpmod"/>  </xs:restriction>  </xs:simpleType>  </xs:element>  <xs:element name="version" type="xs:string">  <xs:annotation>  <xs:documentation>  The version of this named NDM.  </xs:documentation>  </xs:annotation>  </xs:element>  </xs:sequence>  </xs:complexType>  </xs:element>  </xs:sequence>  </xs:complexType>  </xs:element>  </xs:sequence>  </xs:group>  <xs:group name="a1">  <xs:sequence>  <xs:element name="ndm-implementation">  <xs:complexType>  <xs:sequence>  <xs:choice>  <xs:annotation>  <xs:documentation>  This is the choice-construct for configuring specific NDM  implementations on a logical switch.  Specific NDM implementations augment this choice with new  cases with the same name as the corresponding  'available-ndm'  </xs:documentation>  </xs:annotation>  </xs:choice>  </xs:sequence>  </xs:complexType>  </xs:element>  </xs:sequence>  </xs:group> |

### XML Example

This XML Example comes from a specific NDM supporting L2+L3 features as described in [XXX: reference to FAWG document.

|  |
| --- |
| <capable-switch xmlns="urn:onf:of111:config:yang"  xmlns:ndm="urn:opennetworking.org:yang:ndm"  xmlns:l2l3="urn:opennetworking.org:yang:ndm:l2l3">  <logical-switches>  <switch>  <id>LogicalSwitch5</id>  <resources>  <ndm:ndm-implementation>  <l2l3:l2l3>  <l2l3:ingress-vlan-table-size>128</l2l3:ingress-vlan-table-size>  <l2l3:router-mac-table-size>128</l2l3:router-mac-table-size>  <l2l3:l3-table-size>128</l2l3:l3-table-size>  <l2l3:l2-table-size>128</l2l3:l2-table-size>  <l2l3:egress-vlan-table-size>128</l2l3:egress-vlan-table-size>  </l2l3:l2l3>  </ndm:ndm-implementation>  </resources>  </switch>  </logical-switches>  </capable-switch> |

# Binding to NETCONF

Below we specify the requirements and give examples of how the schema specified in section 8 and 2013 is bound to the NETCONF transport protocol.

## Requirements

When implementing the XML schema defined in Section 8 and 2013 the following schemas are required in addition:

* ietf-yang-types.xsd found at <http://www.yang-central.org/modules/xsd/ietf-yang-types.xsd>
* ietf-inet-types.xsd found at <http://www.cablelabs.com/specifications/XSD/ietf-inet-types.xsd>

Those XML schemas define some basic datatypes that are used in the XML schema defined in this document.

A similar set is required when using the YANG model of Appendix B. There you need:

* ietf-yang-types.yang found at <http://www.yang-central.org/modules/yang/ietf-yang-types.yang>
* ietf-inet-types.yang found at <http://www.yang-central.org/modules/yang/ietf-inet-types.yang>

## How the Data Model is Bound to NETCONF

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

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

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

### edit-config

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

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

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

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

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

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

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

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

### get-config

This operation is used 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.

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 which can be found in the description of each individual element.

#### XML Example: get-config

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

### copy-config

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

#### XML Example: copy-config

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

### delete-config

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

#### XML Example: delete-config

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

## RPC error

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

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

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

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

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

1. 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.

1. OF-CONFIG XML Schema

|  |
| --- |
| <?xml version="1.0" encoding="UTF-8"?>  <xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema"  xmlns:yin="urn:ietf:params:xml:schema:yang:yin:1"  targetNamespace="urn:onf:config:yang"  xmlns="urn:onf:config:yang"  elementFormDefault="qualified"  attributeFormDefault="unqualified"  version="2013-10-05"  xml:lang="en"  xmlns:yang="urn:ietf:params:xml:ns:yang:ietf-yang-types"  xmlns:inet="urn:ietf:params:xml:ns:yang:ietf-inet-types"  xmlns:of-config="urn:onf:config:yang">  <xs:import namespace="urn:ietf:params:xml:ns:yang:ietf-yang-types"  schemaLocation="ietf-yang-types.xsd"/>  <xs:import namespace="urn:ietf:params:xml:ns:yang:ietf-inet-types"  schemaLocation="ietf-inet-types.xsd"/>  <xs:annotation>  <xs:documentation>  This schema was generated from the YANG module of-config  by pyang version 1.2.  The schema describes an instance document consisting  of the entire configuration data store, operational  data, rpc operations, and notifications.  This schema can thus NOT be used as-is to  validate NETCONF PDUs.  </xs:documentation>  </xs:annotation>  <xs:annotation>  <xs:documentation>  This module contains a collection of YANG definitions for  configuring OpenFlow datapaths. It is part of the OF-CONFIG  specification.  </xs:documentation>  </xs:annotation>  <!-- YANG typedefs -->  <xs:simpleType name="OFConfigId">  <xs:annotation>  <xs:documentation>  Generic type of an identifier in OF-CONFIG  </xs:documentation>  </xs:annotation>  <xs:restriction base="xs:string">  </xs:restriction>  </xs:simpleType>  <xs:simpleType name="OFConfigurationPointProtocolType">  <xs:annotation>  <xs:documentation>  Possible protocols to connect ot an OF Configuration Point  </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:simpleType name="OFOpenFlowVersionType">  <xs:annotation>  <xs:documentation>  This enumeration contains the all OpenFlow  versions released so far.  </xs:documentation>  </xs:annotation>  <xs:restriction base="xs:string">  <xs:enumeration value="not-applicable"/>  <xs:enumeration value="1.0"/>  <xs:enumeration value="1.0.1"/>  <xs:enumeration value="1.1"/>  <xs:enumeration value="1.2"/>  <xs:enumeration value="1.3"/>  <xs:enumeration value="1.3.1"/>  <xs:enumeration value="1.3.2"/>  </xs:restriction>  </xs:simpleType>  <xs:simpleType name="datapath-id-type">  <xs:annotation>  <xs:documentation>  The datapath-id type represents an OpenFlow  datapath identifier.  </xs:documentation>  </xs:annotation>  <xs:restriction base="xs:string">  <xs:pattern value="[0-9a-fA-F]{2}(:[0-9a-fA-F]{2}){7}"/>  </xs:restriction>  </xs:simpleType>  <xs:simpleType name="OFTenthOfAPercentType">  <xs:annotation>  <xs:documentation>  This type defines a value in tenth of a percent.  </xs:documentation>  </xs:annotation>  <xs:restriction base="xs:unsignedShort">  <xs:minInclusive value="0"/>  <xs:maxInclusive value="1000"/>  </xs:restriction>  </xs:simpleType>  <xs:simpleType name="OFUpDownStateType">  <xs:annotation>  <xs:documentation>  Type to specify state information for a port or a  connection.  </xs:documentation>  </xs:annotation>  <xs:restriction base="xs:string">  <xs:enumeration value="up"/>  <xs:enumeration value="down"/>  </xs:restriction>  </xs:simpleType>  <xs:simpleType name="OFPortRateType">  <xs:annotation>  <xs:documentation>  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 1 Gb can support half or full duplex transmission.  </xs:documentation>  </xs:annotation>  <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="10Gb"/>  <xs:enumeration value="40Gb"/>  <xs:enumeration value="100Gb"/>  <xs:enumeration value="1Tb"/>  <xs:enumeration value="other"/>  </xs:restriction>  </xs:simpleType>  <xs:simpleType name="OFActionType">  <xs:annotation>  <xs:documentation>  The types of actions defined in OpenFlow Switch  Specification versions 1.2, 1.3, and 1.3.1  </xs:documentation>  </xs:annotation>  <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="set-field"/>  </xs:restriction>  </xs:simpleType>  <xs:simpleType name="OFInstructionType">  <xs:annotation>  <xs:documentation>  The types of instructions defined in OpenFlow  Switch Specification versions 1.2, 1.3, and 1.3.1.  </xs:documentation>  </xs:annotation>  <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:simpleType name="OFMatchFieldType">  <xs:annotation>  <xs:documentation>  The types of match field defined in OpenFlow Switch Specification  versions 1.2, 1.3, and 1.3.1.  </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:simpleType name="hex-binary">  <xs:annotation>  <xs:documentation>  Hex binary encoded string  </xs:documentation>  </xs:annotation>  <xs:restriction base="xs:base64Binary">  </xs:restriction>  </xs:simpleType>  <xs:element name="capable-switch">  <xs:annotation>  <xs:documentation>  The OpenFlow Capable Switch serves as the root element for an OpenFlow  configuration. It contains logical switches and resources that  can be assigned to logical switches. It may have relations to  OpenFlow Configuration Points.  </xs:documentation>  </xs:annotation>  <xs:complexType>  <xs:sequence>  <xs:element name="id" type="xs:string">  <xs:annotation>  <xs:documentation>  A unique but locally arbitrary identifier that uniquely identifies a  Capable Switch within the context of potential OpenFlow  Configuration Points. It MUST be persistent across reboots of  the OpenFlow Capable Switch.  </xs:documentation>  </xs:annotation>  </xs:element>  <xs:element name="config-version" minOccurs="0" type="xs:string">  <xs:annotation>  <xs:documentation>  The maximum supported OF-CONFIG version that is supported by the  OpenFlow Capable Switch. For switches implementing this version  of the OF-CONFIG protocol this MUST always be 1.2.  This object can be used to identify the OF-CONFIG version  a capable switch supports beginning with version 1.1.1 of  OF-CONFIG. In addtion the supported version can be  determined by the namespace the OpenFlow Capable Switch  returns to configuration request of an element (like  capable-switch) that is present in all OF-CONFIG versions  specified so far. This is the only possiblity to identify  OF-CONFIG versions prior to OF-CONFIG 1.1.1.  </xs:documentation>  </xs:annotation>  </xs:element>  <xs:element name="configuration-points" minOccurs="0">  <xs:complexType>  <xs:sequence>  <xs:element name="configuration-point" minOccurs="0" maxOccurs="unbounded">  <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:complexType>  <xs:sequence>  <xs:element name="id" type="OFConfigId">  <xs:annotation>  <xs:documentation>  A unique but locally arbitrary identifier that  identifies a Configuration Point within the context of an  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. It  identifies the location of the Configuration Point as a  service resource and MUST include all information necessary  for the OpenFlow Capable Switch to connect to the  Configuration Point or re-connect to it should it become  disconnected. Such information MAY include, for example,  protocol, fully qualified domain name, IP address, port  number, etc.  </xs:documentation>  </xs:annotation>  </xs:element>  <xs:element name="protocol" minOccurs="0" 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:any minOccurs="0" maxOccurs="unbounded"  namespace="##other" processContents="lax"/>  </xs:sequence>  </xs:complexType>  </xs:element>  <xs:any minOccurs="0" maxOccurs="unbounded"  namespace="##other" processContents="lax"/>  </xs:sequence>  </xs:complexType>  <xs:key name="key\_configuration-points\_capable-switch\_configuration-point">  <xs:selector xpath="of-config:configuration-point"/>  <xs:field xpath="of-config:id"/>  </xs:key>  </xs:element>  <xs:element name="resources" minOccurs="0">  <xs:annotation>  <xs:documentation>  A lists containing all resources of the 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.  </xs:documentation>  </xs:annotation>  <xs:complexType>  <xs:sequence>  <xs:element name="port" minOccurs="0" maxOccurs="unbounded">  <xs:annotation>  <xs:documentation>  The list contains all port resources of the OpenFlow Capable Switch.  The element 'resource-id' of OFPortType MUST be unique within  this list.  </xs:documentation>  </xs:annotation>  <xs:complexType>  <xs:sequence>  <xs:element name="resource-id" type="inet:uri">  <xs:annotation>  <xs:documentation>  A unique but locally arbitrary identifier that uniquely identifies an  OpenFlow Port within the context of an OpenFlow Logical Switch.  It MUST be persistent across reboots of the OpenFlow Capable  Switch.  </xs:documentation>  </xs:annotation>  </xs:element>  <xs:element name="number" minOccurs="0" type="xs:unsignedLong">  <xs:annotation>  <xs:documentation>  This number identifies the OpenFlow Port to OpenFlow Controllers. It is  assigned to an OpenFlow Port latest when the OpenFlow Port is  associated with and OpenFlow Logical Switch. If the OpenFlow  Port is associated with an OpenFlow Logical Switch, this element  MUST be unique within the context of the OpenFlow Logical  Switch.    OpenFlow Capable Switch implementations may choose to  assign values to OpenFlow Ports that are unique within the  context of the OpenFlow Logical Switch. These numbers can  be used independent of assignments to OpenFlow Logical  Switches.    Other implementations may assign values to this element  only if the OpenFlow Port is assigned to an OpenFlow  Logical Switch.  </xs:documentation>  </xs:annotation>  </xs:element>  <xs:element name="name" minOccurs="0">  <xs:annotation>  <xs:documentation>  This element assists OpenFlow Controllers in identifying OpenFlow Ports.    This element is not to be set by the OP-CONFIG protocol,  but it is set by the switch implementation. It may be set  at start-up time of an OpenFlow Capable Switch or when the  OpenFlow Port is assigned to an OpenFlow Logical Switch.  It MAY also be not set at all. If this element is set to a  value other than the empty string when being assigned to an  OpenFlow Logical Switch, then the value of this element  MUST be unique within the context of the OpenFlow Logical  Switch.  </xs:documentation>  </xs:annotation>  <xs:simpleType>  <xs:restriction base="xs:string">  <xs:minLength value="1"/>  <xs:maxLength value="16"/>  </xs:restriction>  </xs:simpleType>  </xs:element>  <xs:element name="current-rate" minOccurs="0" type="xs:unsignedInt">  <xs:annotation>  <xs:documentation>  This element indicates the current bit rate of the port. Its values is  to be provided in units of kilobit per second (kbps). This  element is only valid if the element called 'rate' in the  current Port Features has a value of 'other'.  </xs:documentation>  </xs:annotation>  </xs:element>  <xs:element name="max-rate" minOccurs="0" type="xs:unsignedInt">  <xs:annotation>  <xs:documentation>  This element indicates the maximum bit rate of the port. Its values is  to be provided in units of kilobit per second (kbps). This  element is only valid if the element called 'rate' in the  current Port Features has a value of 'other'.  </xs:documentation>  </xs:annotation>  </xs:element>  <xs:element name="configuration" minOccurs="0">  <xs:annotation>  <xs:documentation>  This containter represents the general adminitrative configuration of the  OpenFlow Port.  </xs:documentation>  </xs:annotation>  <xs:complexType>  <xs:sequence>  <xs:element name="admin-state" minOccurs="0" type="OFUpDownStateType">  <xs:annotation>  <xs:documentation>  The administrative state of the port. If true, the port has been  administratively brought down and SHOULD not be used by  OpenFlow.  </xs:documentation>  </xs:annotation>  </xs:element>  <xs:element name="no-receive" minOccurs="0" type="xs:boolean">  <xs:annotation>  <xs:documentation>  If true, packets received at this OpenFlow port SHOULD be dropped.  </xs:documentation>  </xs:annotation>  </xs:element>  <xs:element name="no-forward" minOccurs="0" type="xs:boolean">  <xs:annotation>  <xs:documentation>  If true, packets forwarded to this OpenFlow port SHOULD be dropped.  </xs:documentation>  </xs:annotation>  </xs:element>  <xs:element name="no-packet-in" minOccurs="0" type="xs:boolean">  <xs:annotation>  <xs:documentation>  If true, packets received on that port that generate a table miss should  never trigger a packet-in message to the OpenFlow Controller.  </xs:documentation>  </xs:annotation>  </xs:element>  <xs:any minOccurs="0" maxOccurs="unbounded"  namespace="##other" processContents="lax"/>  </xs:sequence>  </xs:complexType>  </xs:element>  <xs:element name="state" minOccurs="0">  <xs:annotation>  <xs:documentation>  This element represents the general operational state of the OpenFlow  Port.  </xs:documentation>  </xs:annotation>  <xs:complexType>  <xs:sequence>  <xs:element name="oper-state" minOccurs="0" type="OFUpDownStateType">  <xs:annotation>  <xs:documentation>  If the value of this element is 'down', it indicates that there is no  physical link present.  </xs:documentation>  </xs:annotation>  </xs:element>  <xs:element name="blocked" minOccurs="0" type="xs:boolean">  <xs:annotation>  <xs:documentation>  If the value of this element is 'true', it indicates that a switch  protocol outside of OpenFlow, such as 802.1D Spanning Tree, is  preventing the use of this OpenFlow port for OpenFlow  flooding.  </xs:documentation>  </xs:annotation>  </xs:element>  <xs:element name="live" minOccurs="0" type="xs:boolean">  <xs:annotation>  <xs:documentation>  If the value of this element is 'true', it indicates that this OpenFlow  Port is live and can be used for fast failover.  </xs:documentation>  </xs:annotation>  </xs:element>  <xs:any minOccurs="0" maxOccurs="unbounded"  namespace="##other" processContents="lax"/>  </xs:sequence>  </xs:complexType>  </xs:element>  <xs:element name="features" minOccurs="0">  <xs:complexType>  <xs:sequence>  <xs:element name="current" minOccurs="0">  <xs:annotation>  <xs:documentation>  The features (rates, duplex, etc.) of the port, that are currently in  use.  </xs:documentation>  </xs:annotation>  <xs:complexType>  <xs:sequence>  <xs:element name="rate" minOccurs="0" type="OFPortRateType">  <xs:annotation>  <xs:documentation>  The transmission rate that is currently used. The value 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.  </xs:documentation>  </xs:annotation>  </xs:element>  <xs:element name="auto-negotiate" minOccurs="0" type="xs:boolean">  <xs:annotation>  <xs:documentation>  Specifies the administrative state of the forwarding rate  auto-negotiation protocol at this OpenFlow Port.  </xs:documentation>  </xs:annotation>  </xs:element>  <xs:element name="medium" minOccurs="0">  <xs:annotation>  <xs:documentation>  This element MUST indicate a valid physical medium used by the OpenFlow  Port.    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.  </xs:documentation>  </xs:annotation>  <xs:simpleType>  <xs:restriction base="xs:string">  <xs:enumeration value="copper"/>  <xs:enumeration value="fiber"/>  </xs:restriction>  </xs:simpleType>  </xs:element>  <xs:element name="pause" minOccurs="0">  <xs:annotation>  <xs:documentation>  Specifies if pausing of transmission is supported at all and if yes if  it is asymmetric or symmetric.  </xs:documentation>  </xs:annotation>  <xs:simpleType>  <xs:restriction base="xs:string">  <xs:enumeration value="unsupported"/>  <xs:enumeration value="symmetric"/>  <xs:enumeration value="asymmetric"/>  </xs:restriction>  </xs:simpleType>  </xs:element>  <xs:any minOccurs="0" maxOccurs="unbounded"  namespace="##other" processContents="lax"/>  </xs:sequence>  </xs:complexType>  </xs:element>  <xs:element name="advertised" minOccurs="0">  <xs:annotation>  <xs:documentation>  The features (rates, duplex, etc.) of the port, that are advertised to  the peer port.  </xs:documentation>  </xs:annotation>  <xs:complexType>  <xs:sequence>  <xs:element name="rate" minOccurs="1" maxOccurs="unbounded" type="OFPortRateType">  <xs:annotation>  <xs:documentation>  The transmission rate that is supported or advertised. Multiple  transmissions rates are allowed.  </xs:documentation>  </xs:annotation>  </xs:element>  <xs:element name="auto-negotiate" minOccurs="0" type="xs:boolean">  <xs:annotation>  <xs:documentation>  Specifies if auto-negotiation of transmission parameters is enabled for  the port.  </xs:documentation>  </xs:annotation>  </xs:element>  <xs:element name="medium" minOccurs="1" maxOccurs="unbounded">  <xs:annotation>  <xs:documentation>  The transmission medium used by the port. Multiple media are allowed.  </xs:documentation>  </xs:annotation>  <xs:simpleType>  <xs:restriction base="xs:string">  <xs:enumeration value="copper"/>  <xs:enumeration value="fiber"/>  </xs:restriction>  </xs:simpleType>  </xs:element>  <xs:element name="pause">  <xs:annotation>  <xs:documentation>  Specifies if pausing of transmission is supported at all and if yes if  it is asymmetric or symmetric.  </xs:documentation>  </xs:annotation>  <xs:simpleType>  <xs:restriction base="xs:string">  <xs:enumeration value="unsupported"/>  <xs:enumeration value="symmetric"/>  <xs:enumeration value="asymmetric"/>  </xs:restriction>  </xs:simpleType>  </xs:element>  <xs:any minOccurs="0" maxOccurs="unbounded"  namespace="##other" processContents="lax"/>  </xs:sequence>  </xs:complexType>  </xs:element>  <xs:element name="supported" minOccurs="0">  <xs:annotation>  <xs:documentation>  The features (rates, duplex, etc.) of the port, that are supported on  the port.  </xs:documentation>  </xs:annotation>  <xs:complexType>  <xs:sequence>  <xs:element name="rate" minOccurs="1" maxOccurs="unbounded" type="OFPortRateType">  <xs:annotation>  <xs:documentation>  The transmission rate that is supported or advertised. Multiple  transmissions rates are allowed.  </xs:documentation>  </xs:annotation>  </xs:element>  <xs:element name="auto-negotiate" minOccurs="0" type="xs:boolean">  <xs:annotation>  <xs:documentation>  Specifies if auto-negotiation of transmission parameters is enabled for  the port.  </xs:documentation>  </xs:annotation>  </xs:element>  <xs:element name="medium" minOccurs="1" maxOccurs="unbounded">  <xs:annotation>  <xs:documentation>  The transmission medium used by the port. Multiple media are allowed.  </xs:documentation>  </xs:annotation>  <xs:simpleType>  <xs:restriction base="xs:string">  <xs:enumeration value="copper"/>  <xs:enumeration value="fiber"/>  </xs:restriction>  </xs:simpleType>  </xs:element>  <xs:element name="pause">  <xs:annotation>  <xs:documentation>  Specifies if pausing of transmission is supported at all and if yes if  it is asymmetric or symmetric.  </xs:documentation>  </xs:annotation>  <xs:simpleType>  <xs:restriction base="xs:string">  <xs:enumeration value="unsupported"/>  <xs:enumeration value="symmetric"/>  <xs:enumeration value="asymmetric"/>  </xs:restriction>  </xs:simpleType>  </xs:element>  <xs:any minOccurs="0" maxOccurs="unbounded"  namespace="##other" processContents="lax"/>  </xs:sequence>  </xs:complexType>  </xs:element>  <xs:element name="advertised-peer" minOccurs="0">  <xs:annotation>  <xs:documentation>  The features (rates, duplex, etc.) that are currently advertised by the  peer port.  </xs:documentation>  </xs:annotation>  <xs:complexType>  <xs:sequence>  <xs:element name="rate" minOccurs="1" maxOccurs="unbounded" type="OFPortRateType">  <xs:annotation>  <xs:documentation>  The transmission rate that is supported or advertised. Multiple  transmissions rates are allowed.  </xs:documentation>  </xs:annotation>  </xs:element>  <xs:element name="auto-negotiate" minOccurs="0" type="xs:boolean">  <xs:annotation>  <xs:documentation>  Specifies if auto-negotiation of transmission parameters is enabled for  the port.  </xs:documentation>  </xs:annotation>  </xs:element>  <xs:element name="medium" minOccurs="1" maxOccurs="unbounded">  <xs:annotation>  <xs:documentation>  The transmission medium used by the port. Multiple media are allowed.  </xs:documentation>  </xs:annotation>  <xs:simpleType>  <xs:restriction base="xs:string">  <xs:enumeration value="copper"/>  <xs:enumeration value="fiber"/>  </xs:restriction>  </xs:simpleType>  </xs:element>  <xs:element name="pause">  <xs:annotation>  <xs:documentation>  Specifies if pausing of transmission is supported at all and if yes if  it is asymmetric or symmetric.  </xs:documentation>  </xs:annotation>  <xs:simpleType>  <xs:restriction base="xs:string">  <xs:enumeration value="unsupported"/>  <xs:enumeration value="symmetric"/>  <xs:enumeration value="asymmetric"/>  </xs:restriction>  </xs:simpleType>  </xs:element>  <xs:any minOccurs="0" maxOccurs="unbounded"  namespace="##other" processContents="lax"/>  </xs:sequence>  </xs:complexType>  </xs:element>  <xs:any minOccurs="0" maxOccurs="unbounded"  namespace="##other" processContents="lax"/>  </xs:sequence>  </xs:complexType>  </xs:element>  <xs:choice>  <xs:annotation>  <xs:documentation>  Tunnels are modeled as logical ports.  </xs:documentation>  </xs:annotation>  <xs:sequence>  <xs:element name="tunnel" minOccurs="0">  <xs:annotation>  <xs:documentation>  Properties of a basic IP-in-GRE tunnel.  </xs:documentation>  </xs:annotation>  <xs:complexType>  <xs:sequence>  <xs:choice>  <xs:sequence>  <xs:element name="local-endpoint-ipv4-adress" minOccurs="0" type="inet:ipv4-address">  <xs:annotation>  <xs:documentation>  The IPv4 address of the local tunnel endpoint.  </xs:documentation>  </xs:annotation>  </xs:element>  <xs:element name="remote-endpoint-ipv4-adress" minOccurs="0" type="inet:ipv4-address">  <xs:annotation>  <xs:documentation>  The IPv4 address of the remote tunnel endpoint.  </xs:documentation>  </xs:annotation>  </xs:element>  <xs:any minOccurs="0" maxOccurs="unbounded"  namespace="##other" processContents="lax"/>  </xs:sequence>  <xs:sequence>  <xs:element name="local-endpoint-ipv6-adress" minOccurs="0" type="inet:ipv6-address">  <xs:annotation>  <xs:documentation>  The IPv6 address of the local tunnel endpoint.  </xs:documentation>  </xs:annotation>  </xs:element>  <xs:element name="remote-endpoint-ipv6-adress" minOccurs="0" type="inet:ipv6-address">  <xs:annotation>  <xs:documentation>  The IPv6 address of the remote tunnel endpoint.  </xs:documentation>  </xs:annotation>  </xs:element>  <xs:any minOccurs="0" maxOccurs="unbounded"  namespace="##other" processContents="lax"/>  </xs:sequence>  <xs:sequence>  <xs:element name="local-endpoint-mac-adress" minOccurs="0" type="yang:mac-address">  <xs:annotation>  <xs:documentation>  The MAC address of the local tunnel endpoint.  </xs:documentation>  </xs:annotation>  </xs:element>  <xs:element name="remote-endpoint-mac-adress" minOccurs="0" type="yang:mac-address">  <xs:annotation>  <xs:documentation>  The MAC address of the remote tunnel endpoint.  </xs:documentation>  </xs:annotation>  </xs:element>  <xs:any minOccurs="0" maxOccurs="unbounded"  namespace="##other" processContents="lax"/>  </xs:sequence>  <xs:any minOccurs="0" maxOccurs="unbounded"  namespace="##other" processContents="lax"/>  </xs:choice>  <xs:any minOccurs="0" maxOccurs="unbounded"  namespace="##other" processContents="lax"/>  </xs:sequence>  </xs:complexType>  </xs:element>  <xs:any minOccurs="0" maxOccurs="unbounded"  namespace="##other" processContents="lax"/>  </xs:sequence>  <xs:sequence>  <xs:element name="ipgre-tunnel" minOccurs="0">  <xs:annotation>  <xs:documentation>  Properties of a IP-in-GRE tunnel.  </xs:documentation>  </xs:annotation>  <xs:complexType>  <xs:sequence>  <xs:choice>  <xs:sequence>  <xs:element name="local-endpoint-ipv4-adress" minOccurs="0" type="inet:ipv4-address">  <xs:annotation>  <xs:documentation>  The IPv4 address of the local tunnel endpoint.  </xs:documentation>  </xs:annotation>  </xs:element>  <xs:element name="remote-endpoint-ipv4-adress" minOccurs="0" type="inet:ipv4-address">  <xs:annotation>  <xs:documentation>  The IPv4 address of the remote tunnel endpoint.  </xs:documentation>  </xs:annotation>  </xs:element>  <xs:any minOccurs="0" maxOccurs="unbounded"  namespace="##other" processContents="lax"/>  </xs:sequence>  <xs:sequence>  <xs:element name="local-endpoint-ipv6-adress" minOccurs="0" type="inet:ipv6-address">  <xs:annotation>  <xs:documentation>  The IPv6 address of the local tunnel endpoint.  </xs:documentation>  </xs:annotation>  </xs:element>  <xs:element name="remote-endpoint-ipv6-adress" minOccurs="0" type="inet:ipv6-address">  <xs:annotation>  <xs:documentation>  The IPv6 address of the remote tunnel endpoint.  </xs:documentation>  </xs:annotation>  </xs:element>  <xs:any minOccurs="0" maxOccurs="unbounded"  namespace="##other" processContents="lax"/>  </xs:sequence>  <xs:sequence>  <xs:element name="local-endpoint-mac-adress" minOccurs="0" type="yang:mac-address">  <xs:annotation>  <xs:documentation>  The MAC address of the local tunnel endpoint.  </xs:documentation>  </xs:annotation>  </xs:element>  <xs:element name="remote-endpoint-mac-adress" minOccurs="0" type="yang:mac-address">  <xs:annotation>  <xs:documentation>  The MAC address of the remote tunnel endpoint.  </xs:documentation>  </xs:annotation>  </xs:element>  <xs:any minOccurs="0" maxOccurs="unbounded"  namespace="##other" processContents="lax"/>  </xs:sequence>  <xs:any minOccurs="0" maxOccurs="unbounded"  namespace="##other" processContents="lax"/>  </xs:choice>  <xs:element name="checksum-present" minOccurs="0" type="xs:boolean">  <xs:annotation>  <xs:documentation>  Indicates presence of the GRE checksum.  </xs:documentation>  </xs:annotation>  </xs:element>  <xs:element name="key-present" minOccurs="0" type="xs:boolean">  <xs:annotation>  <xs:documentation>  Indicates presence of the GRE key.  </xs:documentation>  </xs:annotation>  </xs:element>  <xs:element name="key" type="xs:unsignedInt">  <xs:annotation>  <xs:documentation>  The (optional) key of the GRE tunnel. It MAY be used to set the  OXM\_OF\_TUNNEL\_ID match field metadata in the OpenFlow protocol  </xs:documentation>  </xs:annotation>  </xs:element>  <xs:element name="sequence-number-present" minOccurs="0" type="xs:boolean">  <xs:annotation>  <xs:documentation>  Indicates presence of the GRE sequence number.  </xs:documentation>  </xs:annotation>  </xs:element>  <xs:any minOccurs="0" maxOccurs="unbounded"  namespace="##other" processContents="lax"/>  </xs:sequence>  </xs:complexType>  </xs:element>  <xs:any minOccurs="0" maxOccurs="unbounded"  namespace="##other" processContents="lax"/>  </xs:sequence>  <xs:sequence>  <xs:element name="vxlan-tunnel" minOccurs="0">  <xs:annotation>  <xs:documentation>  Properties of a VxLAN tunnel.  </xs:documentation>  </xs:annotation>  <xs:complexType>  <xs:sequence>  <xs:choice>  <xs:sequence>  <xs:element name="local-endpoint-ipv4-adress" minOccurs="0" type="inet:ipv4-address">  <xs:annotation>  <xs:documentation>  The IPv4 address of the local tunnel endpoint.  </xs:documentation>  </xs:annotation>  </xs:element>  <xs:element name="remote-endpoint-ipv4-adress" minOccurs="0" type="inet:ipv4-address">  <xs:annotation>  <xs:documentation>  The IPv4 address of the remote tunnel endpoint.  </xs:documentation>  </xs:annotation>  </xs:element>  <xs:any minOccurs="0" maxOccurs="unbounded"  namespace="##other" processContents="lax"/>  </xs:sequence>  <xs:sequence>  <xs:element name="local-endpoint-ipv6-adress" minOccurs="0" type="inet:ipv6-address">  <xs:annotation>  <xs:documentation>  The IPv6 address of the local tunnel endpoint.  </xs:documentation>  </xs:annotation>  </xs:element>  <xs:element name="remote-endpoint-ipv6-adress" minOccurs="0" type="inet:ipv6-address">  <xs:annotation>  <xs:documentation>  The IPv6 address of the remote tunnel endpoint.  </xs:documentation>  </xs:annotation>  </xs:element>  <xs:any minOccurs="0" maxOccurs="unbounded"  namespace="##other" processContents="lax"/>  </xs:sequence>  <xs:sequence>  <xs:element name="local-endpoint-mac-adress" minOccurs="0" type="yang:mac-address">  <xs:annotation>  <xs:documentation>  The MAC address of the local tunnel endpoint.  </xs:documentation>  </xs:annotation>  </xs:element>  <xs:element name="remote-endpoint-mac-adress" minOccurs="0" type="yang:mac-address">  <xs:annotation>  <xs:documentation>  The MAC address of the remote tunnel endpoint.  </xs:documentation>  </xs:annotation>  </xs:element>  <xs:any minOccurs="0" maxOccurs="unbounded"  namespace="##other" processContents="lax"/>  </xs:sequence>  <xs:any minOccurs="0" maxOccurs="unbounded"  namespace="##other" processContents="lax"/>  </xs:choice>  <xs:element name="vni-valid" minOccurs="0" type="xs:boolean">  <xs:annotation>  <xs:documentation>  Indicates how the corresponding flag should be set in packets sent on  the tunnel.  </xs:documentation>  </xs:annotation>  </xs:element>  <xs:element name="vni" minOccurs="0" type="xs:unsignedInt">  <xs:annotation>  <xs:documentation>  Virtual network identifier assigned to all packets sent on the tunnel.  A VxLAN implementation MAY use the this element to set the  OXM\_OF\_TUNNEL\_ID match field metadata in the OpenFlow protocol.  </xs:documentation>  </xs:annotation>  </xs:element>  <xs:element name="vni-multicast-group" minOccurs="0" type="inet:ip-address">  <xs:annotation>  <xs:documentation>  If IP multicast is used to support broadcast on the tunnel this  specifies the corresponding multicast IP address  </xs:documentation>  </xs:annotation>  </xs:element>  <xs:element name="udp-source-port" minOccurs="0" type="inet:port-number">  <xs:annotation>  <xs:documentation>  Specifies the outer UDP source port number. If this element is absent,  the port number MAY be chosen dynamically.  </xs:documentation>  </xs:annotation>  </xs:element>  <xs:element name="udp-dest-port" minOccurs="0" type="inet:port-number">  <xs:annotation>  <xs:documentation>  Specifies the outer UDP destination port number. It SHOULD  be set to 4789, the port number reserved for VxLAN at IANA.  </xs:documentation>  </xs:annotation>  </xs:element>  <xs:element name="udp-checksum" minOccurs="0" type="xs:boolean">  <xs:annotation>  <xs:documentation>  Boolean flag to indicate whether or not the outer UDP checksum should be  set  </xs:documentation>  </xs:annotation>  </xs:element>  <xs:any minOccurs="0" maxOccurs="unbounded"  namespace="##other" processContents="lax"/>  </xs:sequence>  </xs:complexType>  </xs:element>  <xs:any minOccurs="0" maxOccurs="unbounded"  namespace="##other" processContents="lax"/>  </xs:sequence>  <xs:sequence>  <xs:element name="nvgre-tunnel" minOccurs="0">  <xs:annotation>  <xs:documentation>  Properties of a NVGRE tunnel.  </xs:documentation>  </xs:annotation>  <xs:complexType>  <xs:sequence>  <xs:choice>  <xs:sequence>  <xs:element name="local-endpoint-ipv4-adress" minOccurs="0" type="inet:ipv4-address">  <xs:annotation>  <xs:documentation>  The IPv4 address of the local tunnel endpoint.  </xs:documentation>  </xs:annotation>  </xs:element>  <xs:element name="remote-endpoint-ipv4-adress" minOccurs="0" type="inet:ipv4-address">  <xs:annotation>  <xs:documentation>  The IPv4 address of the remote tunnel endpoint.  </xs:documentation>  </xs:annotation>  </xs:element>  <xs:any minOccurs="0" maxOccurs="unbounded"  namespace="##other" processContents="lax"/>  </xs:sequence>  <xs:sequence>  <xs:element name="local-endpoint-ipv6-adress" minOccurs="0" type="inet:ipv6-address">  <xs:annotation>  <xs:documentation>  The IPv6 address of the local tunnel endpoint.  </xs:documentation>  </xs:annotation>  </xs:element>  <xs:element name="remote-endpoint-ipv6-adress" minOccurs="0" type="inet:ipv6-address">  <xs:annotation>  <xs:documentation>  The IPv6 address of the remote tunnel endpoint.  </xs:documentation>  </xs:annotation>  </xs:element>  <xs:any minOccurs="0" maxOccurs="unbounded"  namespace="##other" processContents="lax"/>  </xs:sequence>  <xs:sequence>  <xs:element name="local-endpoint-mac-adress" minOccurs="0" type="yang:mac-address">  <xs:annotation>  <xs:documentation>  The MAC address of the local tunnel endpoint.  </xs:documentation>  </xs:annotation>  </xs:element>  <xs:element name="remote-endpoint-mac-adress" minOccurs="0" type="yang:mac-address">  <xs:annotation>  <xs:documentation>  The MAC address of the remote tunnel endpoint.  </xs:documentation>  </xs:annotation>  </xs:element>  <xs:any minOccurs="0" maxOccurs="unbounded"  namespace="##other" processContents="lax"/>  </xs:sequence>  <xs:any minOccurs="0" maxOccurs="unbounded"  namespace="##other" processContents="lax"/>  </xs:choice>  <xs:element name="vsid" minOccurs="0" type="xs:unsignedInt">  <xs:annotation>  <xs:documentation>  Specifies the virtual subnet id used to identify packets belonging to  the NVGRE virtual layer-2 network (24 bit)  </xs:documentation>  </xs:annotation>  </xs:element>  <xs:element name="flow-id" minOccurs="0" type="xs:unsignedByte">  <xs:annotation>  <xs:documentation>  8-bit value that is used to provide per-flow entropy for flows in the same VSID  </xs:documentation>  </xs:annotation>  </xs:element>  <xs:any minOccurs="0" maxOccurs="unbounded"  namespace="##other" processContents="lax"/>  </xs:sequence>  </xs:complexType>  </xs:element>  <xs:any minOccurs="0" maxOccurs="unbounded"  namespace="##other" processContents="lax"/>  </xs:sequence>  <xs:any minOccurs="0" maxOccurs="unbounded"  namespace="##other" processContents="lax"/>  </xs:choice>  <xs:any minOccurs="0" maxOccurs="unbounded"  namespace="##other" processContents="lax"/>  </xs:sequence>  </xs:complexType>  </xs:element>  <xs:element name="queue" minOccurs="0" maxOccurs="unbounded">  <xs:annotation>  <xs:documentation>  The list contains all queue resources of the OpenFlow Capable Switch.  </xs:documentation>  </xs:annotation>  <xs:complexType>  <xs:sequence>  <xs:element name="resource-id" type="inet:uri">  <xs:annotation>  <xs:documentation>  A unique but locally arbitrary identifier that uniquely identifies an  OpenFlow Port within the context of an OpenFlow Logical Switch.  It MUST be persistent across reboots of the OpenFlow Capable  Switch.  </xs:documentation>  </xs:annotation>  </xs:element>  <xs:element name="id" type="xs:unsignedLong">  <xs:annotation>  <xs:documentation>  This id identifies the OpenFlow Queue to OpenFlow Controllers. It is  assigned to an OpenFlow Queue latest when the OpenFlow Queue is  associated with and OpenFlow Logical Switch. If the OpenFlow  Queue is associated with an OpenFlow Logical Switch, this  element MUST be unique within the context of the OpenFlow  Logical Switch.    OpenFlow Capable Switch implementations may choose to  assign values to OpenFlow Queues that are unique within the  context of the OpenFlow Logical Switch. These id can be  used independent of assignments to OpenFlow Logical  Switches.    Other implementations may assign values to this element  only if the OpenFlow Queue is assigned to an OpenFlow  Logical Switch.  </xs:documentation>  </xs:annotation>  </xs:element>  <xs:element name="port" minOccurs="0">  <xs:annotation>  <xs:documentation>  Reference to port resources in the Capable Switch.    This element associates an OpenFlow Queue with an OpenFlow  Port. If the OpenFlow Queue is associated with an OpenFlow  Logical Switch S and this element is present, then it MUST be  set to the value of element resource-id of an OpenFlow Port  which is associated with the OpenFlow Logical Switch.  </xs:documentation>  </xs:annotation>  <xs:simpleType>  <xs:restriction base="inet:uri">  </xs:restriction>  </xs:simpleType>  </xs:element>  <xs:element name="properties" minOccurs="0">  <xs:annotation>  <xs:documentation>  The queue properties currently configured.  </xs:documentation>  </xs:annotation>  <xs:complexType>  <xs:sequence>  <xs:element name="min-rate" minOccurs="0" type="OFTenthOfAPercentType">  <xs:annotation>  <xs:documentation>  The minimal rate that is reserved for this queue in 1/10 of a percent of  the actual rate.  This element is optional. If not present a min-rate is not  set.  </xs:documentation>  </xs:annotation>  </xs:element>  <xs:element name="max-rate" minOccurs="0" type="OFTenthOfAPercentType">  <xs:annotation>  <xs:documentation>  The maximum rate that is reserved for this queue in 1/10 of a percent of  the actual rate.  This element is optional. If not present the max-rate is not  set.  </xs:documentation>  </xs:annotation>  </xs:element>  <xs:element name="experimenter" minOccurs="0" maxOccurs="unbounded" type="xs:unsignedInt">  <xs:annotation>  <xs:documentation>  A list of experimenter identifiers of queue properties used.  This element is optional.  </xs:documentation>  </xs:annotation>  </xs:element>  <xs:any minOccurs="0" maxOccurs="unbounded"  namespace="##other" processContents="lax"/>  </xs:sequence>  </xs:complexType>  </xs:element>  <xs:any minOccurs="0" maxOccurs="unbounded"  namespace="##other" processContents="lax"/>  </xs:sequence>  </xs:complexType>  </xs:element>  <xs:element name="owned-certificate" minOccurs="0" maxOccurs="unbounded">  <xs:annotation>  <xs:documentation>  The list contains all owned certificate resources of the OpenFlow  Capable Switch.  </xs:documentation>  </xs:annotation>  <xs:complexType>  <xs:sequence>  <xs:element name="resource-id" type="inet:uri">  <xs:annotation>  <xs:documentation>  A unique but locally arbitrary identifier that uniquely identifies an  OpenFlow Port within the context of an OpenFlow Logical Switch.  It MUST be persistent across reboots of the OpenFlow Capable  Switch.  </xs:documentation>  </xs:annotation>  </xs:element>  <xs:element name="certificate" type="xs:string">  <xs:annotation>  <xs:documentation>  An X.509 certificate in DER format base64 encoded.  </xs:documentation>  </xs:annotation>  </xs:element>  <xs:element name="private-key" minOccurs="0">  <xs:annotation>  <xs:documentation>  This element 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.  </xs:documentation>  </xs:annotation>  <xs:complexType>  <xs:sequence>  <xs:choice>  <xs:sequence>  <xs:element name="DSAKeyValue" minOccurs="0">  <xs:complexType>  <xs:sequence>  <xs:element name="P" type="xs:base64Binary">  <xs:annotation>  <xs:documentation>  A prime modulus meeting the requirements of the standard  above  </xs:documentation>  </xs:annotation>  </xs:element>  <xs:element name="Q" type="xs:base64Binary">  <xs:annotation>  <xs:documentation>  An integer in the range 2\*\*159 &lt; Q &lt; 2\*\*160 which is a  prime divisor of P-1  </xs:documentation>  </xs:annotation>  </xs:element>  <xs:element name="J" minOccurs="0" type="xs:base64Binary">  <xs:annotation>  <xs:documentation>  (P - 1) / Q  </xs:documentation>  </xs:annotation>  </xs:element>  <xs:element name="G" minOccurs="0" type="xs:base64Binary">  <xs:annotation>  <xs:documentation>  An integer with certain properties with respect to P and Q  </xs:documentation>  </xs:annotation>  </xs:element>  <xs:element name="Y" type="xs:base64Binary">  <xs:annotation>  <xs:documentation>  G\*\*X mod P (where X is part of the private key and not made  public)  </xs:documentation>  </xs:annotation>  </xs:element>  <xs:element name="Seed" type="xs:base64Binary">  <xs:annotation>  <xs:documentation>  A DSA prime generation seed  </xs:documentation>  </xs:annotation>  </xs:element>  <xs:element name="PgenCounter" type="xs:base64Binary">  <xs:annotation>  <xs:documentation>  A DSA prime generation counter  </xs:documentation>  </xs:annotation>  </xs:element>  <xs:any minOccurs="0" maxOccurs="unbounded"  namespace="##other" processContents="lax"/>  </xs:sequence>  </xs:complexType>  </xs:element>  <xs:any minOccurs="0" maxOccurs="unbounded"  namespace="##other" processContents="lax"/>  </xs:sequence>  <xs:sequence>  <xs:element name="RSAKeyValue" minOccurs="0">  <xs:complexType>  <xs:sequence>  <xs:element name="Modulus" type="xs:base64Binary"/>  <xs:element name="Exponent" type="xs:base64Binary"/>  <xs:any minOccurs="0" maxOccurs="unbounded"  namespace="##other" processContents="lax"/>  </xs:sequence>  </xs:complexType>  </xs:element>  <xs:any minOccurs="0" maxOccurs="unbounded"  namespace="##other" processContents="lax"/>  </xs:sequence>  <xs:any minOccurs="0" maxOccurs="unbounded"  namespace="##other" processContents="lax"/>  </xs:choice>  <xs:any minOccurs="0" maxOccurs="unbounded"  namespace="##other" processContents="lax"/>  </xs:sequence>  </xs:complexType>  </xs:element>  <xs:any minOccurs="0" maxOccurs="unbounded"  namespace="##other" processContents="lax"/>  </xs:sequence>  </xs:complexType>  </xs:element>  <xs:element name="external-certificate" minOccurs="0" maxOccurs="unbounded">  <xs:annotation>  <xs:documentation>  The list contains all external certificate resources of the OpenFlow  Capable Switch.  </xs:documentation>  </xs:annotation>  <xs:complexType>  <xs:sequence>  <xs:element name="resource-id" type="inet:uri">  <xs:annotation>  <xs:documentation>  A unique but locally arbitrary identifier that uniquely identifies an  OpenFlow Port within the context of an OpenFlow Logical Switch.  It MUST be persistent across reboots of the OpenFlow Capable  Switch.  </xs:documentation>  </xs:annotation>  </xs:element>  <xs:element name="certificate" type="xs:string">  <xs:annotation>  <xs:documentation>  An X.509 certificate in DER format base64 encoded.  </xs:documentation>  </xs:annotation>  </xs:element>  <xs:any minOccurs="0" maxOccurs="unbounded"  namespace="##other" processContents="lax"/>  </xs:sequence>  </xs:complexType>  </xs:element>  <xs:element name="flow-table" minOccurs="0" maxOccurs="unbounded">  <xs:annotation>  <xs:documentation>  The list contains all flow table resources of the OpenFlow Capable  Switch.  </xs:documentation>  </xs:annotation>  <xs:complexType>  <xs:sequence>  <xs:element name="resource-id" type="inet:uri">  <xs:annotation>  <xs:documentation>  A unique but locally arbitrary identifier that uniquely identifies an  OpenFlow Port within the context of an OpenFlow Logical Switch.  It MUST be persistent across reboots of the OpenFlow Capable  Switch.  </xs:documentation>  </xs:annotation>  </xs:element>  <xs:element name="max-entries" minOccurs="0" type="xs:unsignedByte">  <xs:annotation>  <xs:documentation>  The maximum number of flow entries supported by the flow table.  </xs:documentation>  </xs:annotation>  </xs:element>  <xs:element name="next-tables" minOccurs="0">  <xs:annotation>  <xs:documentation>  An array of resource-ids of all flow tables that can be directly reached  from this table using the 'goto-table' instruction.  </xs:documentation>  </xs:annotation>  <xs:complexType>  <xs:sequence>  <xs:element name="table-id" minOccurs="0" maxOccurs="unbounded" type="inet:uri"/>  <xs:any minOccurs="0" maxOccurs="unbounded"  namespace="##other" processContents="lax"/>  </xs:sequence>  </xs:complexType>  </xs:element>  <xs:element name="instructions" minOccurs="0">  <xs:annotation>  <xs:documentation>  The list of all instruction types supported by the flow table.  </xs:documentation>  </xs:annotation>  <xs:complexType>  <xs:sequence>  <xs:element name="type" minOccurs="0" maxOccurs="unbounded" type="OFInstructionType"/>  <xs:any minOccurs="0" maxOccurs="unbounded"  namespace="##other" processContents="lax"/>  </xs:sequence>  </xs:complexType>  </xs:element>  <xs:element name="matches" minOccurs="0">  <xs:annotation>  <xs:documentation>  The list of all match types supported by the flow table.  </xs:documentation>  </xs:annotation>  <xs:complexType>  <xs:sequence>  <xs:element name="type" minOccurs="0" maxOccurs="unbounded" type="OFMatchFieldType"/>  <xs:any minOccurs="0" maxOccurs="unbounded"  namespace="##other" processContents="lax"/>  </xs:sequence>  </xs:complexType>  </xs:element>  <xs:element name="write-actions" minOccurs="0">  <xs:annotation>  <xs:documentation>  The list of all write action types supported by the flow table.  </xs:documentation>  </xs:annotation>  <xs:complexType>  <xs:sequence>  <xs:element name="type" minOccurs="0" maxOccurs="unbounded" type="OFActionType"/>  <xs:any minOccurs="0" maxOccurs="unbounded"  namespace="##other" processContents="lax"/>  </xs:sequence>  </xs:complexType>  </xs:element>  <xs:element name="apply-actions" minOccurs="0">  <xs:annotation>  <xs:documentation>  The list of all apply action types supported by the flow table.  </xs:documentation>  </xs:annotation>  <xs:complexType>  <xs:sequence>  <xs:element name="type" minOccurs="0" maxOccurs="unbounded" type="OFActionType"/>  <xs:any minOccurs="0" maxOccurs="unbounded"  namespace="##other" processContents="lax"/>  </xs:sequence>  </xs:complexType>  </xs:element>  <xs:element name="write-setfields" minOccurs="0">  <xs:annotation>  <xs:documentation>  The list of all 'set-field' action types supported by the table using  write actions.  </xs:documentation>  </xs:annotation>  <xs:complexType>  <xs:sequence>  <xs:element name="type" minOccurs="0" maxOccurs="unbounded" type="OFMatchFieldType"/>  <xs:any minOccurs="0" maxOccurs="unbounded"  namespace="##other" processContents="lax"/>  </xs:sequence>  </xs:complexType>  </xs:element>  <xs:element name="apply-setfields" minOccurs="0">  <xs:annotation>  <xs:documentation>  The list of all 'set-field' action types supported by the table using  apply actions.  </xs:documentation>  </xs:annotation>  <xs:complexType>  <xs:sequence>  <xs:element name="type" minOccurs="0" maxOccurs="unbounded" type="OFMatchFieldType"/>  <xs:any minOccurs="0" maxOccurs="unbounded"  namespace="##other" processContents="lax"/>  </xs:sequence>  </xs:complexType>  </xs:element>  <xs:element name="wildcards" minOccurs="0">  <xs:annotation>  <xs:documentation>  The list of all fields for which the table supports wildcarding.  </xs:documentation>  </xs:annotation>  <xs:complexType>  <xs:sequence>  <xs:element name="type" minOccurs="0" maxOccurs="unbounded" type="OFMatchFieldType"/>  <xs:any minOccurs="0" maxOccurs="unbounded"  namespace="##other" processContents="lax"/>  </xs:sequence>  </xs:complexType>  </xs:element>  <xs:element name="metadata-match" minOccurs="0" type="hex-binary">  <xs:annotation>  <xs:documentation>  This element indicates the bits of the metadata field on which the flow  table can match. It is represented as 64-bit integer in  hexadecimal digits([0-9a-fA-F]) format.  </xs:documentation>  </xs:annotation>  </xs:element>  <xs:element name="metadata-write" minOccurs="0" type="hex-binary">  <xs:annotation>  <xs:documentation>  This element indicates the bits of the metadata field on which flow  table can write using the 'write-metadata' instruction. It is  represented as 64-bit integer in hexadecimal digits([0-9a-fA-F])  format.  </xs:documentation>  </xs:annotation>  </xs:element>  <xs:any minOccurs="0" maxOccurs="unbounded"  namespace="##other" processContents="lax"/>  </xs:sequence>  </xs:complexType>  </xs:element>  <xs:any minOccurs="0" maxOccurs="unbounded"  namespace="##other" processContents="lax"/>  </xs:sequence>  </xs:complexType>  <xs:key name="key\_resources\_capable-switch\_port">  <xs:selector xpath="of-config:port"/>  <xs:field xpath="of-config:resource-id"/>  </xs:key>  <xs:key name="key\_resources\_capable-switch\_queue">  <xs:selector xpath="of-config:queue"/>  <xs:field xpath="of-config:resource-id"/>  </xs:key>  <xs:key name="key\_resources\_capable-switch\_owned-certificate">  <xs:selector xpath="of-config:owned-certificate"/>  <xs:field xpath="of-config:resource-id"/>  </xs:key>  <xs:key name="key\_resources\_capable-switch\_external-certificate">  <xs:selector xpath="of-config:external-certificate"/>  <xs:field xpath="of-config:resource-id"/>  </xs:key>  <xs:key name="key\_resources\_capable-switch\_flow-table">  <xs:selector xpath="of-config:flow-table"/>  <xs:field xpath="of-config:resource-id"/>  </xs:key>  </xs:element>  <xs:element name="logical-switches" minOccurs="0">  <xs:annotation>  <xs:documentation>  This element contains a list of all OpenFlow Logical Switches available  at the OpenFlow Capable Switch.  </xs:documentation>  </xs:annotation>  <xs:complexType>  <xs:sequence>  <xs:element name="switch" minOccurs="0" maxOccurs="unbounded">  <xs:annotation>  <xs:documentation>  The list of all OpenFlow Logical Switches on the OpenFlow Capable  Switch.  </xs:documentation>  </xs:annotation>  <xs:complexType>  <xs:sequence>  <xs:element name="id" type="OFConfigId">  <xs:annotation>  <xs:documentation>  A unique but locally arbitrary identifier that  identifies a Logical Switch within the context of an  OpenFlow Capable Switch. It MUST be persistent across  reboots of the OpenFlow Capable Switch.  </xs:documentation>  </xs:annotation>  </xs:element>  <xs:element name="capabilities" minOccurs="0">  <xs:annotation>  <xs:documentation>  This element contains all capability items that  an OpenFlow Logical Switch MAY implement.  </xs:documentation>  </xs:annotation>  <xs:complexType>  <xs:sequence>  <xs:element name="max-buffered-packets" minOccurs="0" type="xs:unsignedInt">  <xs:annotation>  <xs:documentation>  The maximum number of packets the logical switch  can buffer when sending packets to the controller using  packet-in messages.  </xs:documentation>  </xs:annotation>  </xs:element>  <xs:element name="max-tables" minOccurs="0" type="xs:unsignedByte">  <xs:annotation>  <xs:documentation>  The number of flow tables supported by the logical switch.  </xs:documentation>  </xs:annotation>  </xs:element>  <xs:element name="max-ports" minOccurs="0" type="xs:unsignedInt">  <xs:annotation>  <xs:documentation>  The number of flow tables supported by the logical switch.  </xs:documentation>  </xs:annotation>  </xs:element>  <xs:element name="flow-statistics" minOccurs="0" type="xs:boolean">  <xs:annotation>  <xs:documentation>  Specifies if the logical switch supports flow statistics.  </xs:documentation>  </xs:annotation>  </xs:element>  <xs:element name="table-statistics" minOccurs="0" type="xs:boolean">  <xs:annotation>  <xs:documentation>  Specifies if the logical switch supports table statistics.  </xs:documentation>  </xs:annotation>  </xs:element>  <xs:element name="port-statistics" minOccurs="0" type="xs:boolean">  <xs:annotation>  <xs:documentation>  Specifies if the logical switch supports port statistics.  </xs:documentation>  </xs:annotation>  </xs:element>  <xs:element name="group-statistics" minOccurs="0" type="xs:boolean">  <xs:annotation>  <xs:documentation>  Specifies if the logical switch supports group statistics.  </xs:documentation>  </xs:annotation>  </xs:element>  <xs:element name="queue-statistics" minOccurs="0" type="xs:boolean">  <xs:annotation>  <xs:documentation>  Specifies if the logical switch supports queue statistics.  </xs:documentation>  </xs:annotation>  </xs:element>  <xs:element name="reassemble-ip-fragments" minOccurs="0" type="xs:boolean">  <xs:annotation>  <xs:documentation>  Specifies if the logical switch supports reassemble IP fragments.  </xs:documentation>  </xs:annotation>  </xs:element>  <xs:element name="block-looping-ports" minOccurs="0" 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.  </xs:documentation>  </xs:annotation>  </xs:element>  <xs:element name="reserved-port-types" minOccurs="0">  <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.  </xs:documentation>  </xs:annotation>  <xs:complexType>  <xs:sequence>  <xs:element name="type" minOccurs="0" maxOccurs="unbounded">  <xs:simpleType>  <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="normal"/>  <xs:enumeration value="flood"/>  </xs:restriction>  </xs:simpleType>  </xs:element>  <xs:any minOccurs="0" maxOccurs="unbounded"  namespace="##other" processContents="lax"/>  </xs:sequence>  </xs:complexType>  </xs:element>  <xs:element name="group-types" minOccurs="0">  <xs:annotation>  <xs:documentation>  Specify the group types supported by the logical switch.  </xs:documentation>  </xs:annotation>  <xs:complexType>  <xs:sequence>  <xs:element name="type" minOccurs="0" maxOccurs="unbounded">  <xs:simpleType>  <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:element>  <xs:any minOccurs="0" maxOccurs="unbounded"  namespace="##other" processContents="lax"/>  </xs:sequence>  </xs:complexType>  </xs:element>  <xs:element name="group-capabilities" minOccurs="0">  <xs:annotation>  <xs:documentation>  Specify the group capabilities supported by the logical switch.  </xs:documentation>  </xs:annotation>  <xs:complexType>  <xs:sequence>  <xs:element name="capability" minOccurs="0" maxOccurs="unbounded">  <xs:simpleType>  <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:element>  <xs:any minOccurs="0" maxOccurs="unbounded"  namespace="##other" processContents="lax"/>  </xs:sequence>  </xs:complexType>  </xs:element>  <xs:element name="action-types" minOccurs="0">  <xs:annotation>  <xs:documentation>  Specify the action types supported by the logical switch.  </xs:documentation>  </xs:annotation>  <xs:complexType>  <xs:sequence>  <xs:element name="type" minOccurs="0" maxOccurs="unbounded" type="OFActionType"/>  <xs:any minOccurs="0" maxOccurs="unbounded"  namespace="##other" processContents="lax"/>  </xs:sequence>  </xs:complexType>  </xs:element>  <xs:element name="instruction-types" minOccurs="0">  <xs:annotation>  <xs:documentation>  Specify the instruction types supported by the logical switch.  </xs:documentation>  </xs:annotation>  <xs:complexType>  <xs:sequence>  <xs:element name="type" minOccurs="0" maxOccurs="unbounded" type="OFInstructionType"/>  <xs:any minOccurs="0" maxOccurs="unbounded"  namespace="##other" processContents="lax"/>  </xs:sequence>  </xs:complexType>  </xs:element>  <xs:any minOccurs="0" maxOccurs="unbounded"  namespace="##other" processContents="lax"/>  </xs:sequence>  </xs:complexType>  </xs:element>  <xs:element name="datapath-id" type="datapath-id-type">  <xs:annotation>  <xs:documentation>  The datapath identifier of the Logical Switch  that uniquely identifies this Logical Switch within the  context of all OpenFlow Controllers associated with the  OpenFlow Logical Switch. The datapath identifier is a  string value that MUST be formatted as a sequence of 8  2-digit hexadecimal numbers that are separated by colons,  for example, '01:23:45:67:89:ab:cd:ef'. When processing a  datapath identifier, the case of the decimal digits MUST be  ignored.  </xs:documentation>  </xs:annotation>  </xs:element>  <xs:element name="enabled" minOccurs="0" type="xs:boolean">  <xs:annotation>  <xs:documentation>  This element indicates 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.  </xs:documentation>  </xs:annotation>  </xs:element>  <xs:element name="check-controller-certificate" minOccurs="0" type="xs:boolean">  <xs:annotation>  <xs:documentation>  This element indicates the behavior of the  OpenFlow Logical Switch when connecting to an OpenFlow  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 &lt;protocol&gt; set to  'TLS', only if the controller provides a certificate that  can be verified with one of the certificates stored in the  list called external-certificates in the OpenFlow Capable  Switch.    If a certificate cannot be validated, the OpenFlow Logical  Switch MUST terminate communication with the corresponding  OpenFlow Controller, MUST NOT conduct any OpenFlow  processing on requests of this OpenFlow controller, and  SHOULD NOT further utilize any computational or network  resources of for dealing with this connection.    If set to value 'true', the OpenFlow Logical Switch MUST  NOT connect to any OpenFlow Controller that does not  provide a certificate. This implies that it cannot connect  to an OpenFlow controller that has the value of element  protocol set to 'TCP'. Only connections with protocol 'TLS'  are possible in this case.  </xs:documentation>  </xs:annotation>  </xs:element>  <xs:element name="lost-connection-behavior" minOccurs="0">  <xs:annotation>  <xs:documentation>  This element indicates the the behavior of the  OpenFlow Logical Switch in case it loses contact with all  OpenFlow Controllers. There are two alternative modes in  such a case: fails secure mode and fail standalone mode as  defined by the OpenFlow protocol specification version 1.2,  section 6.4. These are the only allowed values for this  element. Default is the fail secure mode.  </xs:documentation>  </xs:annotation>  <xs:simpleType>  <xs:restriction base="xs:string">  <xs:enumeration value="failSecureMode"/>  <xs:enumeration value="failStandaloneMode"/>  </xs:restriction>  </xs:simpleType>  </xs:element>  <xs:element name="controllers" minOccurs="0">  <xs:annotation>  <xs:documentation>  The list of controllers for this Logical switch.  </xs:documentation>  </xs:annotation>  <xs:complexType>  <xs:sequence>  <xs:element name="controller" minOccurs="0" maxOccurs="unbounded">  <xs:annotation>  <xs:documentation>  The list of OpenFlow Controllers that are  assigned to the OpenFlow Logical Switch. The switch MUST  NOT connect to any OpenFlow Controller that is not  contained in this list.  </xs:documentation>  </xs:annotation>  <xs:complexType>  <xs:sequence>  <xs:element name="id" type="OFConfigId">  <xs:annotation>  <xs:documentation>  A unique but locally arbitrary identifier that uniquely identifies an  OpenFlow Controller within the context of an OpenFlow Capable  Switch. It MUST be persistent across reboots of the OpenFlow  Capable Switch.  </xs:documentation>  </xs:annotation>  </xs:element>  <xs:element name="role" minOccurs="0">  <xs:annotation>  <xs:documentation>  This element indicates the role of the OpenFlow Controller. Semantics of  these roles are specified in the OpenFlow specifications 1.0 -  1.3.1. It is RECOMMENDED that the roles of controllers are not  configured by OF-CONFIG 1.1.1 but determined using the OpenFlow  protocol. OpenFlow Controllers configured by OF-CONFIG 1.1.1  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.  </xs:documentation>  </xs:annotation>  <xs:simpleType>  <xs:restriction base="xs:string">  <xs:enumeration value="master"/>  <xs:enumeration value="slave"/>  <xs:enumeration value="equal"/>  </xs:restriction>  </xs:simpleType>  </xs:element>  <xs:element name="ip-address" type="inet:ip-address">  <xs:annotation>  <xs:documentation>  The IP address of the OpenFlow Controller. This IP address is used by  the OpenFlow Logical Switch when connecting to the OpenFlow  Controller.  </xs:documentation>  </xs:annotation>  </xs:element>  <xs:element name="port" minOccurs="0" type="inet:port-number">  <xs:annotation>  <xs:documentation>  The TCP port number at the OpenFlow Controller. This port number is  used by the OpenFlow Logical Switch when connecting to the  OpenFlow Controller using TCP or TLS. The default value is  6633.  </xs:documentation>  </xs:annotation>  </xs:element>  <xs:element name="local-ip-address" minOccurs="0" type="inet:ip-address">  <xs:annotation>  <xs:documentation>  The local IP address of the OpenFlow Logical Switch when connecting to  this OpenFlow Controller. It is the source IP address of  packets sent to this OpenFlow Controller. If present, this  element overrides any default IP address.    This element is optional. Attempts to set this element to an IP  address that cannot be used by the OpenFlow Logical Switch MUST  result in an 'bad-element' error with type 'application'. The  &lt;error-info&gt; element MUST contain the name of this element in  the &lt;bad-element&gt; element.  </xs:documentation>  </xs:annotation>  </xs:element>  <xs:element name="local-port" minOccurs="0" type="inet:port-number">  <xs:annotation>  <xs:documentation>  The local TCP port number of the OpenFlow Logical Switch when connecting  to this OpenFlow Controller. It is the source TCP port number  of packets sent to this OpenFlow Controller. If this element is  not present, then the port number is chosen arbitrarily by the  OpenFlow Logical Switch.    This element is optional. Attempts to set this element to a  port number that cannot be used by the OpenFlow Logical  Switch MUST result in an 'bad-element' error with type  'application'. The &lt;error-info&gt; element MUST contain the  name of this element in the &lt;bad-element&gt; element.  </xs:documentation>  </xs:annotation>  </xs:element>  <xs:element name="protocol" minOccurs="0">  <xs:annotation>  <xs:documentation>  The default protocol that the OpenFlow Logical Switch uses to connect to  this OpenFlow Controller.  </xs:documentation>  </xs:annotation>  <xs:simpleType>  <xs:restriction base="xs:string">  <xs:enumeration value="tcp"/>  <xs:enumeration value="tls"/>  </xs:restriction>  </xs:simpleType>  </xs:element>  <xs:element name="state" minOccurs="0">  <xs:annotation>  <xs:documentation>  This container holds connection state information that indicate the  connection state of the OpenFlow Logical Switch and the OpenFlow  protocol version used for the connection.  </xs:documentation>  </xs:annotation>  <xs:complexType>  <xs:sequence>  <xs:element name="connection-state" minOccurs="0" type="OFUpDownStateType">  <xs:annotation>  <xs:documentation>  This object indicates the connections state of the OpenFlow Logical  Switch to this controller.  </xs:documentation>  </xs:annotation>  </xs:element>  <xs:element name="current-version" minOccurs="0" type="OFOpenFlowVersionType">  <xs:annotation>  <xs:documentation>  This object indicates the version of the OpenFlow protocol used between  the OpenFlow Logical Switch and this Controller. If element  connection-state has value 'up', then this element indicates  the actual version in use. If element connection-state has  value 'down', then this element indicates the version number  of the last established connection with this OpenFlow  Controller. The value of this element MAY be persistent  across reboots of the OpenFlow Logical Switch in such a case.  If element connection-state has value 'down'and there is no  information about previous connections to this OpenFlow  controller, then this element is not present or has the value  '0'.  </xs:documentation>  </xs:annotation>  </xs:element>  <xs:element name="supported-versions" minOccurs="0" maxOccurs="unbounded" type="OFOpenFlowVersionType">  <xs:annotation>  <xs:documentation>  This list of elements includes one entry for each OpenFlow protocol  version that this OpenFlow controller supports. It SHOULD  contain all  </xs:documentation>  </xs:annotation>  </xs:element>  <xs:element name="local-ip-address-in-use" minOccurs="0" type="inet:ip-address">  <xs:annotation>  <xs:documentation>  The local IP address of the OpenFlow Logical Switch when connecting to  this OpenFlow Controller. It is the source IP address of  packets sent to this OpenFlow Controller. If present, this  element overrides any default IP address.  </xs:documentation>  </xs:annotation>  </xs:element>  <xs:element name="local-port-in-use" minOccurs="0" type="inet:port-number">  <xs:annotation>  <xs:documentation>  The local TCP port number of the OpenFlow Logical Switch. If element  connection-state has value 'up', then this element indicates  the actual port number in use. If element connection-state  has value 'down', then this element indicates the port number  used for the last attempt to establish a connection with this  OpenFlow Controller.??? When connecting to this OpenFlow  Controller, it is the source TCP port number of packets sent  to this OpenFlow Controller. If this element has its defaqult  value 0, then port number is chosen arbitrarily by the  OpenFlow Logical Switch.  </xs:documentation>  </xs:annotation>  </xs:element>  <xs:any minOccurs="0" maxOccurs="unbounded"  namespace="##other" processContents="lax"/>  </xs:sequence>  </xs:complexType>  </xs:element>  <xs:any minOccurs="0" maxOccurs="unbounded"  namespace="##other" processContents="lax"/>  </xs:sequence>  </xs:complexType>  </xs:element>  <xs:any minOccurs="0" maxOccurs="unbounded"  namespace="##other" processContents="lax"/>  </xs:sequence>  </xs:complexType>  <xs:key name="key\_controllers\_switch\_logical-switches\_capable-switch\_controller">  <xs:selector xpath="of-config:controller"/>  <xs:field xpath="of-config:id"/>  </xs:key>  </xs:element>  <xs:element name="resources" minOccurs="0">  <xs:annotation>  <xs:documentation>  The list of identifiers of all resources of the  OpenFlow Capable Switch that the OpenFlow Logical Switch  has exclusive or non-exclusive access to. A resource is  identified by the value of its resource-identifier element.  For each resource identifier value in this list, there MUST  be an element with a matching resource identifier value in  the resources list of the OpenFlow Capable Switch.    Identifiers of this list are contained in elements  indicating the type of resource: 'port', 'queue',  'certificate', or 'flow-table'. Depending on the type,  different constraints apply. These are specified in  separate descriptions per type.  </xs:documentation>  </xs:annotation>  <xs:complexType>  <xs:sequence>  <xs:element name="port" minOccurs="0" maxOccurs="unbounded">  <xs:annotation>  <xs:documentation>  A resource identifier of a port of the OpenFlow Capable  Switch that the OpenFlow Logical Switch has exclusive access to.  Elements in this list MUST be unique. This means each  port element can only be referenced once.  </xs:documentation>  </xs:annotation>  <xs:simpleType>  <xs:restriction base="inet:uri">  </xs:restriction>  </xs:simpleType>  </xs:element>  <xs:element name="queue" minOccurs="0" maxOccurs="unbounded">  <xs:annotation>  <xs:documentation>  A resource identifier of a queue of the OpenFlow Capable Switch  that the OpenFlow Logical Switch has exclusive access to.  Elements in this list MUST be unique. This means each  queue element can only be referenced once.  </xs:documentation>  </xs:annotation>  <xs:simpleType>  <xs:restriction base="inet:uri">  </xs:restriction>  </xs:simpleType>  </xs:element>  <xs:element name="certificate" minOccurs="0">  <xs:annotation>  <xs:documentation>  The resource identifier of the owned certificate in the OpenFlow Capable  Switch that the OpenFlow Logical Switch uses to identify  itself. This element MUST NOT occur more than once in an  OpenFlow Logical Switch's resource list.    If no such element is in an OpenFlow Logical Switch's resource  list, then the OpenFlow Logical Switch does not authenticate  itself towards an OpenFloe Controller with a certificate. If  this element is present, then the OpenFlow Logical Switch MUST  provide this certificate for authentication to an OpenFlow  Controller when setting up a TLS connection.    For TCP connections this element is irrelevant.  </xs:documentation>  </xs:annotation>  <xs:simpleType>  <xs:restriction base="inet:uri">  </xs:restriction>  </xs:simpleType>  </xs:element>  <xs:element name="flow-table" minOccurs="0" maxOccurs="unbounded">  <xs:annotation>  <xs:documentation>  A resource identifier of a flow table of the OpenFlow Capable  Switch that the OpenFlow Logical Switch has exclusive access to.  Elements in this list MUST be unique. This means each  flow-table element can only be referenced once.  </xs:documentation>  </xs:annotation>  <xs:simpleType>  <xs:restriction base="inet:uri">  </xs:restriction>  </xs:simpleType>  </xs:element>  <xs:any minOccurs="0" maxOccurs="unbounded"  namespace="##other" processContents="lax"/>  </xs:sequence>  </xs:complexType>  </xs:element>  <xs:any minOccurs="0" maxOccurs="unbounded"  namespace="##other" processContents="lax"/>  </xs:sequence>  </xs:complexType>  </xs:element>  <xs:any minOccurs="0" maxOccurs="unbounded"  namespace="##other" processContents="lax"/>  </xs:sequence>  </xs:complexType>  <xs:key name="key\_logical-switches\_capable-switch\_switch">  <xs:selector xpath="of-config:switch"/>  <xs:field xpath="of-config:id"/>  </xs:key>  </xs:element>  <xs:any minOccurs="0" maxOccurs="unbounded"  namespace="##other" processContents="lax"/>  </xs:sequence>  </xs:complexType>  </xs:element>  </xs:schema> |