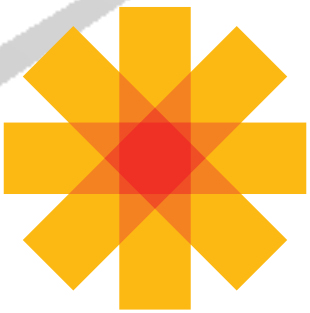
OPEN
DAYLIGHT

OpenDaylight User Guide

()



OPEN
DAYLIGHT

OpenDaylight User Guide

Table of Contents

List of Figures

List of Tables

Part I. Getting Started with Opendaylight

This first part of the user guide covers the basic user operations of the OpenDaylight Release using the generic base functionality.

1. OpenDaylight Controller Overview

The OpenDaylight controller is JVM software and can be run from any operating system and hardware as long as it supports Java. The controller is an implementation of the Software Defined Network (SDN) concept and makes use of the following tools:

- **Maven**: OpenDaylight uses Maven for easier build automation. Maven uses pom.xml (Project Object Model) to script the dependencies between bundle and also to describe what bundles to load and start.
- **OSGi**: This framework is the back-end of OpenDaylight as it allows dynamically loading bundles and packages JAR files, and binding bundles together for exchanging information.
- **JAVA interfaces**: Java interfaces are used for event listening, specifications, and forming patterns. This is the main way in which specific bundles implement call-back functions for events and also to indicate awareness of specific state.
- **REST APIs**: These are northbound APIs such as topology manager, host tracker, flow programmer, static routing, and so on.

The controller exposes open northbound APIs which are used by applications. The OSGi framework and bidirectional REST are supported for the northbound APIs. The OSGi framework is used for applications that run in the same address space as the controller while the REST (web-based) API is used for applications that do not run in the same address space (or even the same system) as the controller. The business logic and algorithms reside in the applications. These applications use the controller to gather network intelligence, run its algorithm to do analytics, and then orchestrate the new rules throughout the network. On the southbound, multiple protocols are supported as plugins, e.g. OpenFlow 1.0, OpenFlow 1.3, BGP-LS, and so on. The OpenDaylight controller starts with an OpenFlow 1.0 southbound plugin. Other OpenDaylight contributors begin adding to the controller code. These modules are linked dynamically into a **Service Abstraction Layer** (SAL).

The SAL exposes services to which the modules north of it are written. The SAL figures out how to fulfill the requested service irrespective of the underlying protocol used between the controller and the network devices. This provides investment protection to the applications as OpenFlow and other protocols evolve over time. For the controller to control devices in its domain, it needs to know about the devices, their capabilities, reachability, and so on. This information is stored and managed by the **Topology Manager**. The other components like ARP handler, Host Tracker, Device Manager, and Switch Manager help in generating the topology database for the Topology Manager.

For a more detailed overview of the OpenDaylight controller, see the *OpenDaylight Developer Guide*.

2. Using the OpenDaylight User Interface (DLUX)

Table of Contents

This section introduces you to the OpenDaylight User Experience (DLUX) application. DLUX is an openflow network management application for Opendaylight controller. For detailed information about Opendaylight overview and architecture, see Opendaylight Overview, Opendaylight Architecture.

OpendaylightController has two interfaces namely:

- AD-SAL
- MD-SAL

Controller receives information from the various interdependent modules through the SAL services. For more information about the SAL services available, see SAL Services. DLUX also uses the SAL services to obtain network related information and use it to provide network management capabilities.

Getting Started with DLUX

You can either use DLUX as a stand-alone plug-in or integrate with the Opendaylight controller.

To install DLUX as a standalone application see OpenDaylight DLUX:Setup and Run.

To integrate with Opendaylight Controller you must enable DLUX Karaf feature. You can enable adsal, md sal and various other bundles within Karaf depending on the features you would like to access using DLUX. Each feature can be enabled or disabled separately.

Important

Ensure that you have created a topology and enabled MD-SAL feature in the Karaf distribution before you use DLUX for network management.

For more information about enabling the Karaf features for DLUX, see OpenDaylight DLUX:DLUX Karaf Feature.

Logging In

To log in to DLUX, after installing the application:

1. Open a browser and enter the login URL. If you have installed DLUX as a stand-alone, then the login URL is http://localhost:9000/DLUX/index.html. However if you have deployed DLUX with Karaf, then the login URL is http://\<your IP\>:8181/dlux/index.html. NOTE: Ensure that you use the port applicable for the DLUX installation type. **local host** is the IP address of the machine where the application us installed.

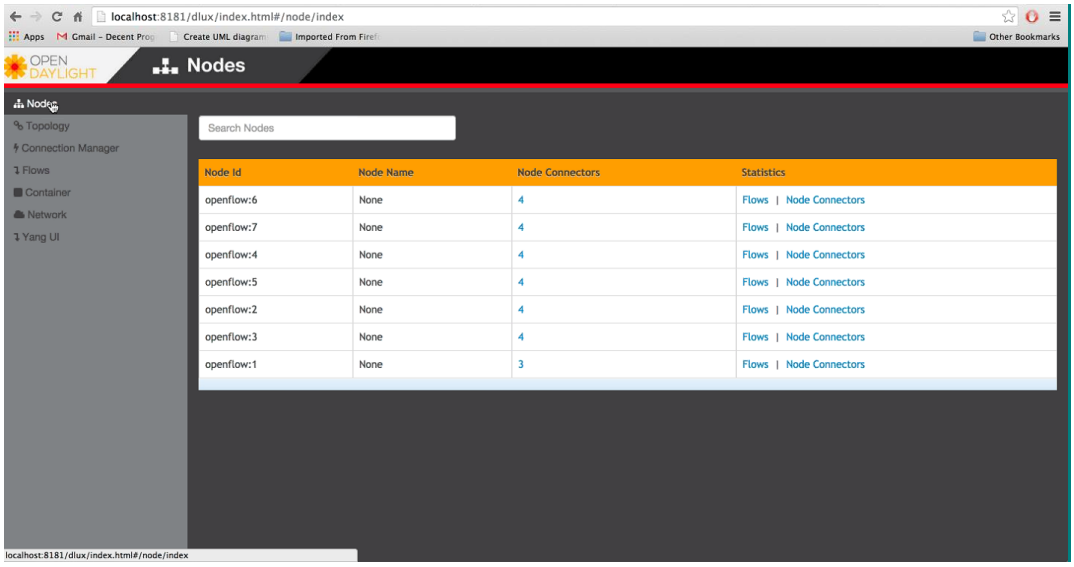
2. Login to the application with user ID and password credentials as **admin**. NOTE: admin is the only user type available for DLUX in this release.

Working with DLUX

After you login to DLUX, you will see all the modules that are available for DLUX in the left pane. However the modules disappear if the features are not enabled in the Karaf distribution.

To get a complete DLUX feature list, install restconf, odl l2 switch, and switch while you start the DLUX distribution. For more information about enabling features on DLUX, see OpenDaylight DLUX:DLUX Karaf Feature.

Figure 2.1. DLUX Modules



Modules that use the MD SAL based apis are :

- Nodes
- Yang UI
- Topology

Modules that use the AD SAL based apis are:

- Connection manager
- Container
- Network
- Flows

Note

DLUX enables only those modules, whose APIs are responding. If you enable just the MD-SAL in beginning and then start dlux, only MD-SAL related tabs will be visible. While using the GUI if you enable AD-SAL karaf features, those tabs will appear automatically.

To view features that are enabled:

1. Right click on the DLUX page.
2. Select **Inspect Element** and then click **Network**. A table that contains the list of features and if they are available in the DLUX distribution. The features that are not enabled is highlighted with red font and has status **404 Not Found**.

Viewing Network Statistics

The **Nodes** module on the left pane enables you to view the network statistics and port information for the switches in the network.

To use the **Nodes** module:

1. Select **Nodes** on the left pane. The right pane displays atable that lists all the nodes, node connectors and the statistics.
2. Enter a node ID in the **Search Nodes** tab to search by node connectors.
3. Click on the **Node Connector** number to view details such as port ID, port name, number of ports per switch, MAC Address, and so on.
4. Click **Flows** in the Statistics column to view Flow Table Statistics for the particular node like table ID, packet match, active flows and so on.
5. Click **Node Connectors** to view Node Connector Statistics for the particular node ID.

Viewing Network Topology

The Topology tab displays a graphical representation of network topology created.

Note

DLUX UI does not provide ability to add topology information. The Topology should be created using an open flow plugin. Controller stores this information in the database and displays on the DLUX page, when the you connect to the controller using openflow.

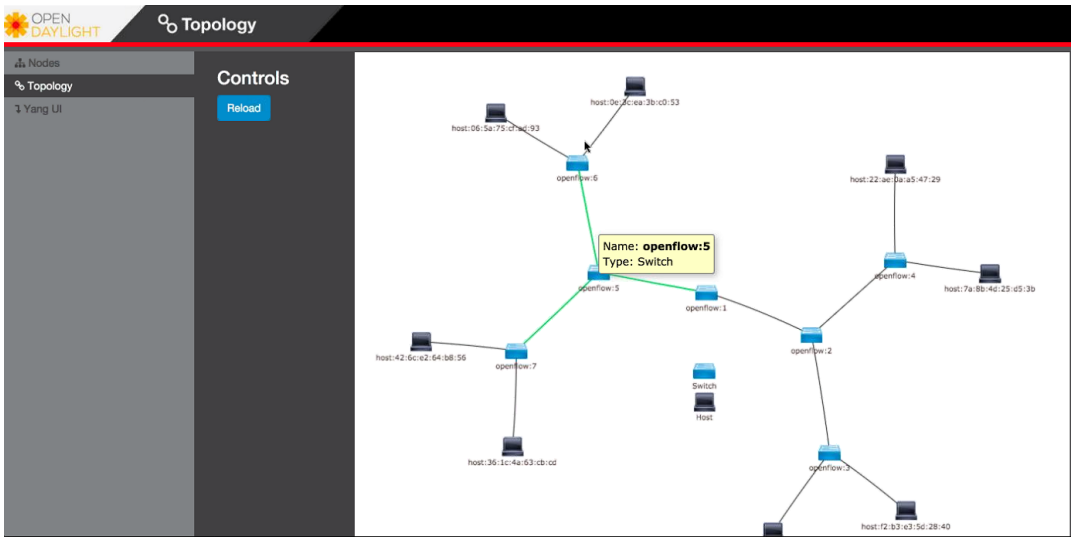
To view network topology:

1. Select **Topology** on the left pane. You will view the graphical representation on the right pane. In the diagram blue boxes represent the switches, the black represents the hosts available, and lines represents how switches are connected.

2. Hover your mouse on hosts,links, or switches to view source and destination ports.

3. Zoom in and zoom out using mouse scroll to verify topology for huge topologies.

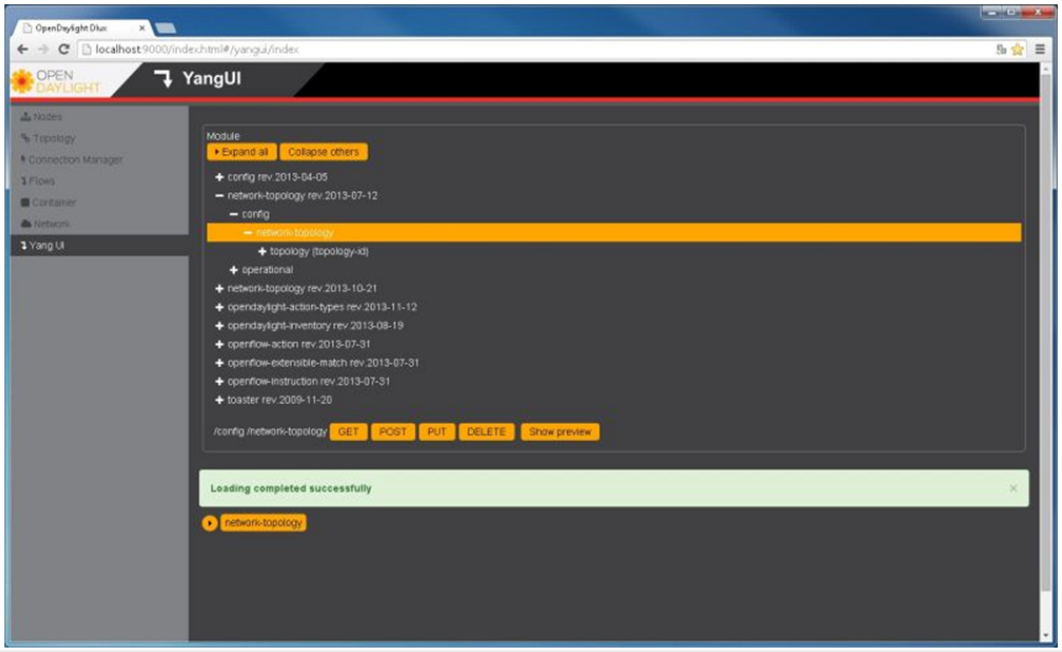
Figure 2.2. Topology Module



Interacting with the Open Daylight Controller (ODL)

The **Yang UI** module enables you to interact with the ODL. For more information about Yang Tools, see https://wiki.opendaylight.org/view/YANG_Tools:Main [YANG_Tools].

Figure 2.3. Yang UI



To use Yang UI:

1. Select **Yang UI** on the left pane. The right pane is divided in two parts.

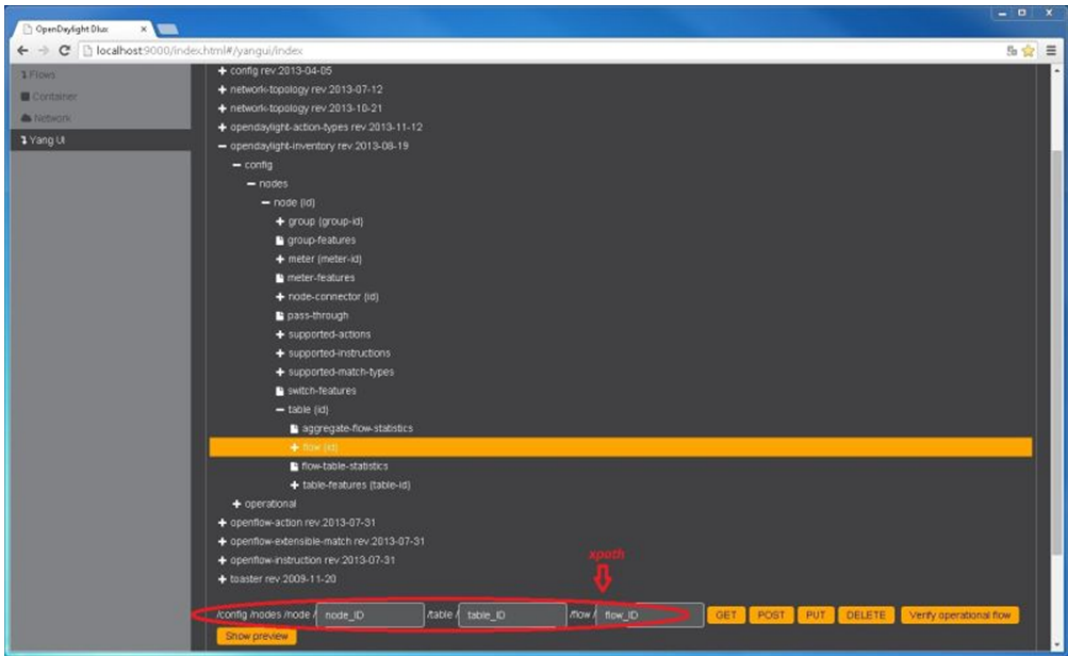
2. The top part displays a tree of APIs and subAPIs and buttons to call possible functions (GET, POST, PUT, DELETE, ...). Not every subAPIs can call every function. For example, subAPIs “operational” have GET functionality only. Inputs can be filled from ODL when existing data from ODL is displayed or can be filled by user on the page and sent to ODL. Buttons under the API tree are variable. It depends on subAPI specifications. Common buttons are:

   - GET to get data from ODL,

   - PUT and POST for sending data to ODL for saving

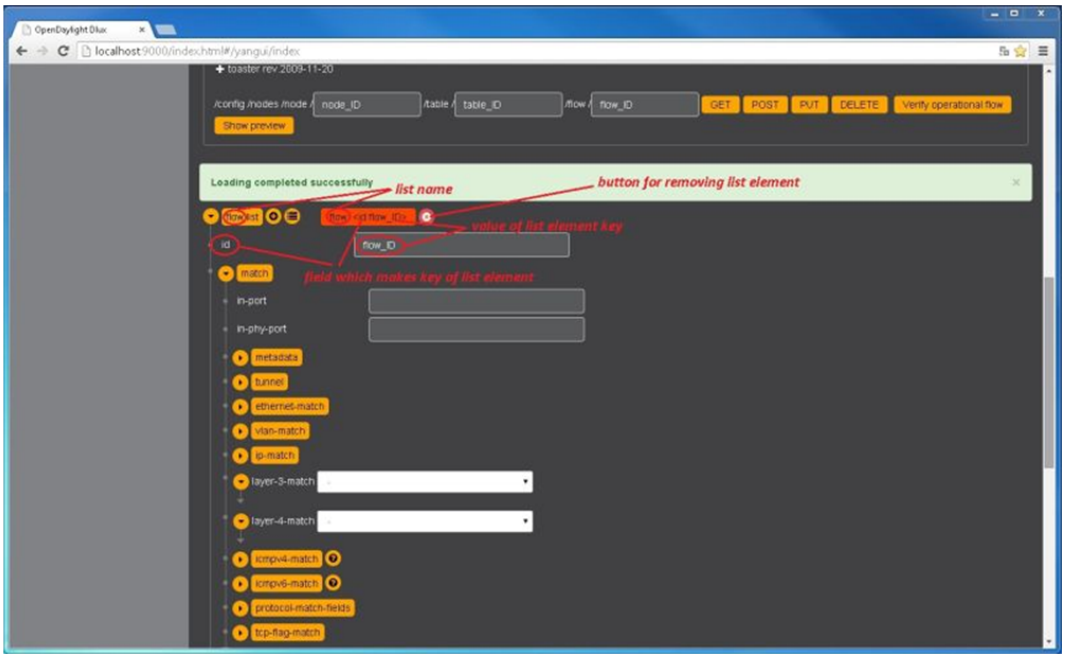
   - DELETE for sending data to ODL for deleting. You must specify the xpath for all these operations. This path is displayed in the same row before buttons and it can include text inputs for specific path elements identifiers.

Figure 2.4. Yang API Specification



3. The bottom part of the right pane displays inputs according to the chosen subAPI. Every subAPI is represented by list elements of list statement. It is possible to have a many list elements of one list. + For example, a device can store multiple flows. In this case “flow” is name of the list and every list element is different by a key value. List element of list can obtain other lists. Every list element has a list name, a key name and its value, and a button for removing this list element. Usually the key of the list statement obtains an ID. Inputs can be filled from ODL using GET button from xpath part, or can be filled by user on the page and sent to ODL.

Figure 2.5. Yang UI API Specification



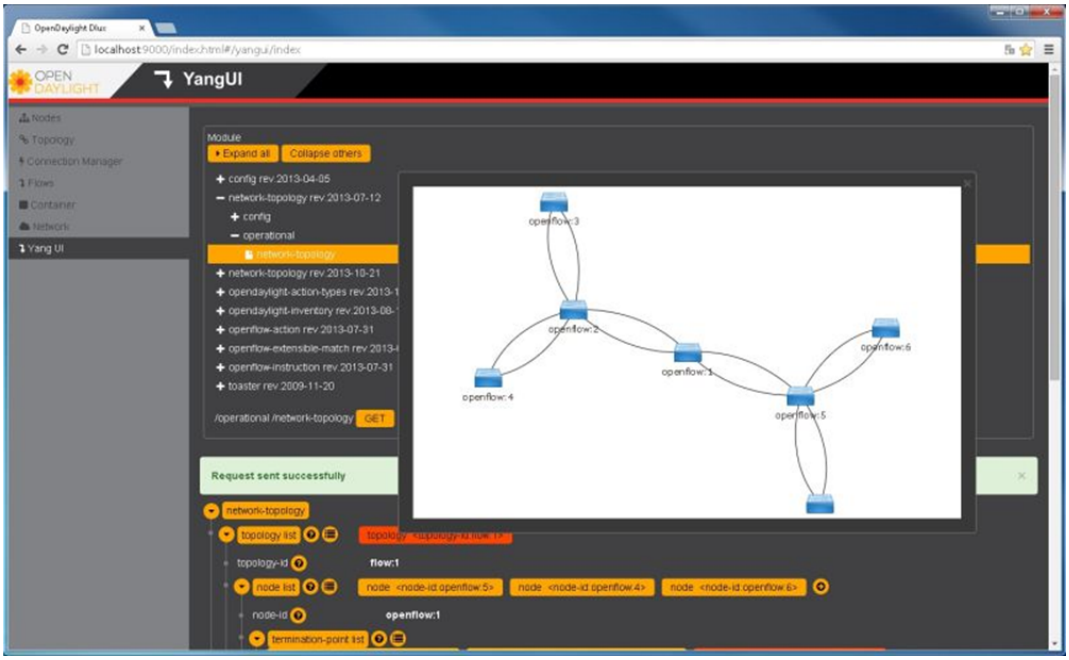
4. Click **Show Preview** button under API tree to display request that will be sent to ODL. A pane is displayed on the right side with text of request when some input is filled.

Displaying Topology on the Yang UI

To display topology:

1. Select subAPI network-topology <topology revision number> # operational # network-topology.
2. Get data from ODL by clicking on the “GET” button.
3. Click **Display Topology**.

Figure 2.6. DLUX Yang Topology



Configuring List Elements on the Yang UI

The list is displayed like tree structure with possibility to expand or collapse by the arrow before name of the list. To configure list elements on the Yang UI:

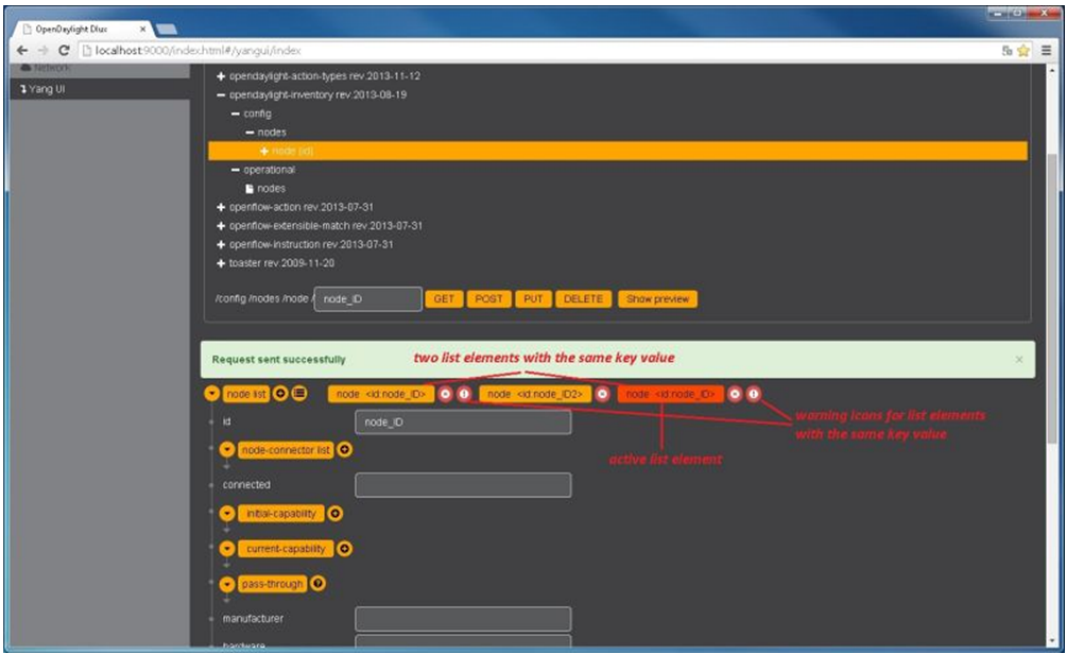
1. To add a new list element with empty inputs use the plus icon-button + that is provided after list name. When some list element is added, button with his name and key value is displayed.

2. To remove several list elements, use the **X** button that is provided after every list element.

   DLUX List Elements. image::dlux-yang-list elements.png[DLUX list elements,width=500]

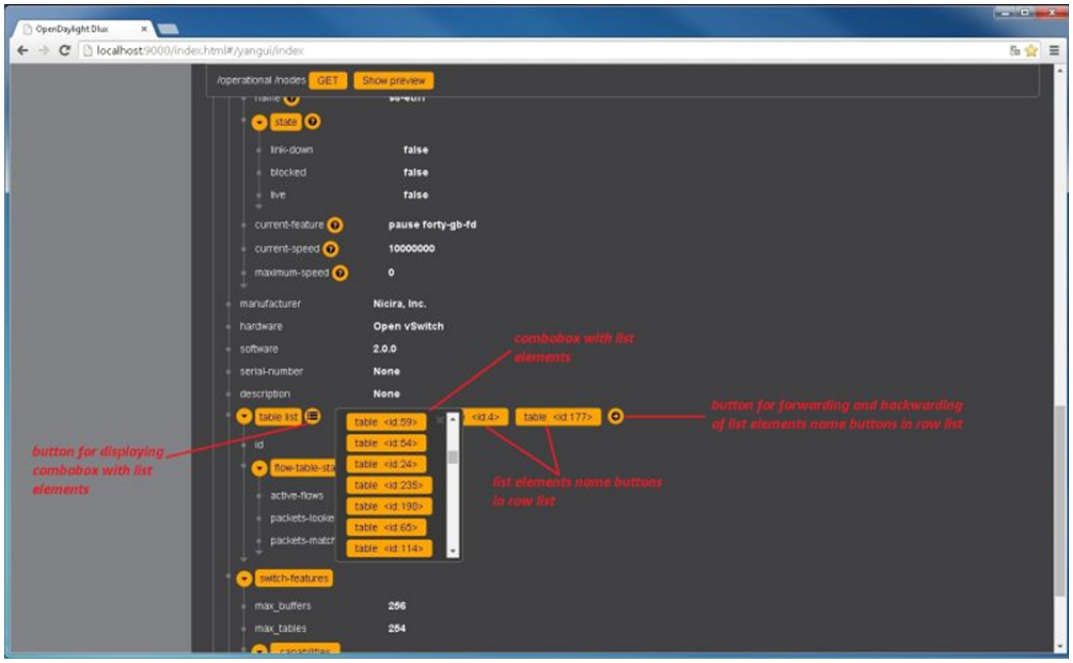
3. Key of list is one or more inputs, which are used like identifier of list element. All list elements in one list must have different key values. If some elements has the same key values, the new warning icon **!** is displayed near their name buttons.

Figure 2.7. DLUX List Warnings



4. When the list obtains at least one list element, after + icon is icon for selecting the list element displayed. You can choose one of them by clicking the icon. The name button of the list element and name buttons of its neighbours will be displayed in the row list. You can can forward or backward row list of list elements name buttons by clicking on the arrow button on the end of row.

Figure 2.8. DLUX List Button1



3. Running XSQL Console Commands and Queries

Table of Contents

XSQL Overview

XSQL is an XML-based query language that describes simple stored procedures which parse XML data, query or update database tables, and compose XML output. It allows you to query tree models as if they were a sequential database. For example, you could run a query that lists all of the ports configured on a particular module and their attributes.

The following sections will cover the XSQL installation process, supported XSQL commands, and the proper way to structure queries.

Installing XSQL

Before you can run commands from the XSQL console, you must first install XSQL onto your system:

1. Navigate to the directory in which you unzipped the OpenDaylight source files.
2. Start Karaf: **./karaf**
3. Install XSQL: **feature:install odl-mdsal-xsql**

XSQL Console Commands

When entering a command in the XSQL console, structure it as follows: **odl:xsql** <*XSQL command*>

The following table describes the commands supported in the OpenDaylight Helium release.

Table 3.1. Supported XSQL Console Commands

Command	Description
r	Repeats the last command you executed.
list vtables	Lists the schema node containers that are currently installed. Whenever an OpenDaylight module is installed, its YANG model is placed in the Schema Context. At that

	point, the XSQL receives a notification, confirms that the module's YANG model resides in the Schema Context, and then maps the model to XSQL by setting up the necessary vtables and vfields. This command is useful when you need to determine vtable information for a query.
list vfields <*vtable name*>	Lists the vfields present in a specific vtable. This command is useful when you need to determine vfields information for a query.
jdbc <*ip address*>	When the ODL server is behind a firewall, and the JDBC client cannot connect to the JDBC server, run this command to start the client as if it was a server and establish a connection.
exit	Closes the console.
tocsv	Enables/disables the forwarding of query output as a .csv file.
filename <*filename*>	Specifies the .tocsv file to which query data is exported. If you do not specify a value for this option when the toccsv option is enabled, the filename for the query data file is generated automatically.

XSQL Queries

Using the information provided by the **list vtables** and **list vfields** <*vtable name*> commands, you can run a query to extract information that meets the criteria you specify. Any query you run should be structured as follows:

select <*vfields you want to search for, separated by a comma and a space*> **from** <*vtables you want to search in, separated by a comma and a space*> **where** <*criteria*> **'**<*criteria operator*>**';**

For example, say you want to search the nodes/node.ID field in the nodes/node-connector table and find every instance of the Hardware-Address object that contains *BA* in its text string. To do so, you would enter the following query: **Select nodes/node.ID from nodes/node-connector where Hardware-Address like '%BA%';**

The following criteria operators are supported:

Table 3.2. Supported XSQL Query Criteria Operators

Criteria Operators	Description
=	Lists results that equal the value you specify.
!=	Lists results that do not equal the value you specify.
like	Lists results that contain the substring you specify. For example, if you specify **like** %**BC**%, every string that contains that particular substring is displayed.
<	Lists results that are less than the value you specify.
>	Lists results that are more than the value you specify.
and	Lists results that match both values you specify.
or	Lists results that match either of the two values you specify.
>=	Lists results that are more than or equal to the value you specify.
#	Lists results that are less than or equal to the value you specify.

is null	Lists results for which no value is assigned.
not null	Lists results for which any value is assigned.
skip	Use this operator to list matching results from a child node, even if its parent node does not meet the specified criteria. See the following example for more information.

Example: skip Criteria Operator

Say you are looking at the following structure and want to determine all of the ports that belong to a YY type module:

- Network Element 1
  - Module 1, Type XX
    - Module 1.1, Type YY
      - Port 1
      - Port 2
  - Module 2, Type YY
    - Port 1
    - Port 2

If you specify **Module.Type='YY'** in your query criteria, the ports associated with module 1.1 will not be returned since its parent module is type XX. Instead, enter **Module.Type='YY' or skip Module!='YY'**. This tells XSQL to disregard any parent module data that does not meet the type YY criteria and collect results for any matching child modules. In this example, you are instructing the query to skip module 1 and collect the relevant data from module 1.1.

4. Setting Up Clustering on an OpenDaylight Controller

Table of Contents

Clustering Overview

Clustering is a mechanism that enables multiple processes and programs to work together as one entity. For example, when you go to google.com and search for something, it may seem like your search request is processed by only one web server. In reality, your search request is processed by thousands of web servers connected in a cluster. Similarly, you can have multiple instances of the OpenDaylight controller working together as one entity. There are a number of uses for clustering:

- Scaling: If you have multiple controllers running, you can potentially do more work with or store more data on those controllers if they are clustered. You can also break up your data into smaller chunks (known as shards) and either distribute that data across the cluster or perform certain operations on certain members of the cluster.
- High Availability: If you have multiple controllers running and one of them crashes, you would still have the other instances working and available.
- Data Persistence: You will not lose any data gathered by your controller after a manual restart or a crash.

The following sections describe how to set up clustering on both individual and multiple OpenDaylight controllers.

Single Node Clustering

To enable clustering on a single OpenDaylight controller, do the following:

1. Download and unzip a base controller distribution. You must use the new openflow plugin, so download a distribution where the new openflow plugin is either the default or can be enabled.
2. Navigate to the <*Karaf-distribution-location*>/bin directory.
3. Run Karaf: **./karaf**
4. Install the clustering feature: **feature:install odl-mdsal-clustering**
5. If you are using the integration distribution of Karaf, you should also install the open flow plugin flow services: **feature:install odl-openflowplugin-flow-services**

6. Install the Jolokia bundle: **install -s mvn:org.jolokia/jolokia-osgi/1.1.5**

After enabling the DistributedDataStore feature in a single instance, you can access the following features:

- Data Sharding: The in-memory MD-SAL tree is broken up into a number of smaller sub-trees (inventory, topology, and default).
- Data Persistence: All of the data available on defined data shards is stored on a disk. By restarting the controller, you can use the persisted data to reinstate those shards to their previous state.

Multiple Node Clustering

The following sections describe how to set up multiple node clusters in OpenDaylight.

Deployment Considerations

Here is some information to keep in mind when you implement clustering:

- When setting up a cluster with multiple nodes, we recommend that you do so with a minimum of three machines. You can set up with a cluster with just two nodes. However, if one of those two nodes go down, the controller will no longer be operational.
- Every device that belongs to a cluster needs to have an identifier. For this purpose, OpenDaylight uses the node's role. After you define the first node's role as *member-1* in the akka.conf file, OpenDaylight uses *member-1* to identify that node.
- Data shards are used to house all or a certain segment of a module's data. For example, one shard can contain all of a module's inventory data while another shard contains all of it's topology data. If you do not specify a module in the modules.conf file and do not specify a shard in module-shards.conf, then (by default) all the data is places onto the default shard (which must also be defined in module-shards.conf file). Each shard has replicas configured, and the module-shards.conf file is where you can specify where these replicas reside.
- Say you have a three node cluster on which HA is enabled. A replica of every defined data shard must be running on all three cluster nodes. This is because OpenDaylight's clustering implementation requires a majority of the defined shard replicas to be running in order to function. If you only define data shard replicas on two of the cluster nodes and one of those nodes goes down, the corresponding data shards will not function.
- If you have a three node cluster and have defined replicas for a data shard on each of those nodes, that shard will still function even if only two of the cluster nodes are running. Note that if one of those two nodes go down, your controller will no longer be operational.

What considerations need to be made when setting the seed nodes for each member? Why are we referring to multiple seed nodes when you set only one IP address? Can you set multiple seed nodes for functional testing?

We recommend that you have multiple seed nodes configured. After a cluster member is started, it sends a message to all of its seed nodes. The cluster member then sends a join command to the first seed node that responds. If none of its seed nodes reply, the cluster member repeats this process until it successfully establishes a connection or it is shutdown.

What happens after one node becomes unreachable? Do the other two nodes function normally? When the first node reconnects, does it automatically synchronize with the other nodes?

After a node becomes unreachable, it remains down for configurable period of time (10 seconds, by default). Once a node goes down, you need to restart it so that it can rejoin the cluster. Once a restarted node joins a cluster, it will synchronize with the lead node automatically.

Can you run a two node cluster for functional testing?

For functional testing, yes. For HA testing, you need to run all three nodes.

Setting Up a Multiple Node Cluster

To run an OpenDaylight controller in a three node cluster, do the following:

1. Determine the three machines that will make up the cluster and copy the controller distribution to each of those machines.
2. Unzip the controller distribution.
3. Navigate to the <*Karaf-distribution-location*>/bin directory.
4. Run Karaf: **./karaf**
5. Install the clustering feature: **feature:install odl-mdsal-clustering**

Note

To run clustering, you must install the odl-mdsal-clustering feature on each of your nodes.

1. If you are using the integration distribution of Karaf, you should also install the open flow plugin flow services: **feature:install odl-openflowplugin-flow-services**
2. Install the Jolokia bundle: **install -s mvn:org.jolokia/jolokia-osgi/1.1.5**
3. On each node, open the following .conf files:
   - configuration/initial/akka.conf
   - configuration/initial/module-shards.conf
4. In each configuration file, make the following changes:
   a. Find every instance of the following lines and replace *127.0.0.1* with the hostname or IP address of the machine on which the controller will run:

```
netty.tcp {
  hostname = "127.0.0.1"
```

Note

The value you need to specify will be different for each node in the cluster.

a. Find the following lines and replace *127.0.0.1* with the hostname or IP address of any of the machines that will be part of the cluster:

```
cluster {
  seed-nodes = ["akka.tcp://opendaylight-cluster-data@127.0.0.1:2550"]
```

b. Find the following section and specify the role for each member node. For example, you could assign the first node with the *member-1* role, the second node with the *member-2* role, and the third node with the *member-3* role.

```
roles = [
  "member-1"
]
```

c. Open the configuration/initial/module-shards.conf file and update the items listed in the following section so that the replicas match roles defined in this host's akka.conf file.

```
replicas = [
    "member-1"
]
```

For reference, view a sample akka.conf file here: https://gist.github.com/moizr/88f4bd4ac2b03cfa45f0

a. Run the following commands on each of your cluster's nodes:

- **JAVA_MAX_MEM=4G JAVA_MAX_PERM_MEM=512m ./karaf**
- **JAVA_MAX_MEM=4G JAVA_MAX_PERM_MEM=512m ./karaf**
- **JAVA_MAX_MEM=4G JAVA_MAX_PERM_MEM=512m ./karaf**

The OpenDaylight controller can now run in a three node cluster. Use any of the three member nodes to access the data residing in the datastore.

Say you want to view information about shard designated as *member-1* on a node. To do so, query the shard's data by making the following HTTP request:

GET http://<host>:8181/jolokia/read/org.opendaylight.controller:Category=Shards,name=member-1-shard-inventory-config,type=DistributedConfigDatastore

Note

If prompted, enter *admin* as both the username and password.

This request should return the following information:


```
{
    "timestamp": 1410524741,
    "status": 200,
    "request": {
    "mbean": "org.opendaylight.controller:Category=Shards,name=member-1-shard-
inventory-config,type=DistributedConfigDatastore",
    "type": "read"
    },
    "value": {
    "ReadWriteTransactionCount": 0,
    "LastLogIndex": -1,
    "MaxNotificationMgrListenerQueueSize": 1000,
    "ReadOnlyTransactionCount": 0,
    "LastLogTerm": -1,
    "CommitIndex": -1,
    "CurrentTerm": 1,
    "FailedReadTransactionsCount": 0,
    "Leader": "member-1-shard-inventory-config",
    "ShardName": "member-1-shard-inventory-config",
    "DataStoreExecutorStats": {
    "activeThreadCount": 0,
    "largestQueueSize": 0,
    "currentThreadPoolSize": 1,
    "maxThreadPoolSize": 1,
    "totalTaskCount": 1,
    "largestThreadPoolSize": 1,
    "currentQueueSize": 0,
    "completedTaskCount": 1,
    "rejectedTaskCount": 0,
    "maxQueueSize": 5000
    },
    "FailedTransactionsCount": 0,
    "CommittedTransactionsCount": 0,
    "NotificationMgrExecutorStats": {
    "activeThreadCount": 0,
    "largestQueueSize": 0,
    "currentThreadPoolSize": 0,
    "maxThreadPoolSize": 20,
    "totalTaskCount": 0,
    "largestThreadPoolSize": 0,
    "currentQueueSize": 0,
    "completedTaskCount": 0,
    "rejectedTaskCount": 0,
    "maxQueueSize": 1000
    },
    "LastApplied": -1,
    "AbortTransactionsCount": 0,
    "WriteOnlyTransactionCount": 0,
    "LastCommittedTransactionTime": "1969-12-31 16:00:00.000",
    "RaftState": "Leader",
    "CurrentNotificationMgrListenerQueueStats": []
    }
}
```


The key thing here is the name of the shard. Shard names are structured as follows:

<member-name>-shard-*<shard-name-as-per-configuration>*-*<store-type>*

Here are a couple sample data short names:

- member-1-shard-topology-config
- member-2-shard-default-operational

Enabling HA on a Multiple Node Cluster

To enable HA in a three node cluster:

1. Open the configuration/initial/module-shards.conf file on each cluster node.
2. Add *member-2* and *member-3* to the replica list for each data shard.
3. Restart all of the nodes. The nodes should automatically sync up with member-1. After some time, the cluster should be ready for operation.

When HA is enabled, you must have at least three replicas of every shard. Each node's configuration files should look something like this:

```
module-shards = [
    {
        name = "default"
        shards = [
            {
                name="default"
                replicas = [
                    "member-1",
                    "member-2",
                    "member-3"
                ]
            }
        ]
    },
    {
        name = "topology"
        shards = [
            {
                name="topology"
                replicas = [
                    "member-1",
                    "member-2",
                    "member-3"
                ]
            }
        ]
    },
    {
        name = "inventory"
        shards = [
            {
                name="inventory"
                replicas = [
                    "member-1",
                    "member-2",
                    "member-3"
```

```
                ]
            }
        ]
    },
    {
         name = "toaster"
         shards = [
             {
                 name="toaster"
                 replicas = [
                    "member-1",
                    "member-2",
                    "member-3"
                 ]
             }
         ]
    }
]
```

When HA is enabled on multiple nodes, shards will replicate the data for those nodes. Whenever the lead replica on a data shard is brought down, another replica takes its place. As a result, the cluster should remain available. To determine which replica is acting as the lead on a data shard, make an HTTP request to obtain the information for a data shard on any of the nodes. The resulting information will indicate which replica is acting as the lead.

Part II. Addons

This second part of the user guide covers project specific usage instructions.

5. BGP LS PCEP

Table of Contents

BGP LS

OpenDaylight comes pre-configured in the installation. You can find it in the opendaylight/configuration/initial directory and it consists of two files:

31-bgp.xml, which defines the basic parser and RIB support. Unless you need to add a new AFI/SAFI, you should keep this file as is.

41-bgp-example.xml, which contains a sample configuration which needs to be customized to your deployment.

Currently the configuration for BGP peer is ignored in the configuration, to prevent the client from starting with default configuration. Therefore the first step is to uncomment ALL the commented parts in this file.

1. Adjust values for initial BGP Open message

```
<module>
    <type>prefix:rib-impl</type>
    <name>example-bgp-rib</name>
    <rib-id>example-bgp-rib</rib-id>
    <local-as>64496</local-as>         // Our AS number, we use this in best
 path selection
    <bgp-id>192.0.2.2</bgp-id>         // Our BGP identifier, we use this in
 best path selection
```

2. Specify IP address of your BGP speaker

```
<module>
    <type
 xmlns:prefix="urn:opendaylight:params:xml:ns:yang:controller:bgp:rib:impl">
        prefix:bgp-peer
    </type>
    <name>example-bgp-peer</name>
    <host>192.0.2.1</host>                          // IP address or hostname
 of the speaker
    <holdtimer>180</holdtimer>
```

You can also add more BGP peers with different instance name and hostname.

```
<module>
    <type
 xmlns:prefix="urn:opendaylight:params:xml:ns:yang:controller:bgp:rib:impl">
        prefix:bgp-peer
    </type>
    <name>example-bgp-peer2</name>
    <host>192.0.2.2</host>
```

```
    <holdtimer>180</holdtimer>
```

1. Configure connection attributes (all in milliseconds)

```
<module>
   <type
 xmlns:prefix="urn:opendaylight:params:xml:ns:yang:controller:reconnectstrategy">
        prefix:timed-reconnect-strategy
   </type>
   <name>example-reconnect-strategy</name>
   <min-sleep>1000</min-sleep>             // Minimum sleep time in between
 reconnect tries
   <max-sleep>180000</max-sleep>           // Maximum sleep time in between
 reconnect tries
   <sleep-factor>2.00</sleep-factor>       // Power factor of the sleep time
 between reconnect tries
   <connect-time>5000</connect-time>       // How long we should wait for the
 TCP connect attempt, overrides default connection timeout dictated by TCP
 retransmits
   <executor>
       <type
 xmlns:netty="urn:opendaylight:params:xml:ns:yang:controller:netty">
           netty:netty-event-executor
       </type>
       <name>global-event-executor</name>
   </executor>
</module>
```

BGP speaker configuration

Previous entries addressed the configuration of a BGP connection initiated by ODL. ODL also supports BGP Speaker functionality and accepts incoming BGP connections.

The configuration of BGP speaker is located in 41-bgp-example.xml.

```
<module>
   <type xmlns:prefix=
"urn:opendaylight:params:xml:ns:yang:controller:bgp:rib:impl">
       prefix:bgp-peer-acceptor
   </type>
   <name>bgp-peer-server</name>
   <!--Default parameters-->
    <!--<binding-address>0.0.0.0</binding-address>-->
    <!--<binding-port>179</binding-port>-->
   <bgp-dispatcher>
       <type xmlns:prefix=
"urn:opendaylight:params:xml:ns:yang:controller:bgp:rib:impl">
           prefix:bgp-dispatcher
       </type>
       <name>global-bgp-dispatcher</name>
   </bgp-dispatcher>
   <!--Drops or accepts incoming BGP connection, every BGP Peer that should be
 accepted needs to be added to this registry-->
   <peer-registry>
       <type xmlns:prefix=
"urn:opendaylight:params:xml:ns:yang:controller:bgp:rib:impl">
           prefix:bgp-peer-registry
       </type>
       <name>global-bgp-peer-registry</name>
   </peer-registry>
```

```
</module>
```

1. Changing speaker configuration

   - Changing binding address: Uncomment tag binding-address and change the address to e.g. 127.0.0.1. The default binding address is 0.0.0.0.
   - Changing binding port: Uncomment tag binding-port and change the port to e.g. 1790. The default binding port is 179 as specified in BGP RFC.

2. Configuring incoming BGP connections

By default, the **BGP speaker drops all BGP connections from unknown BGP peers**. The decision is made in component bgp-peer-registry that is injected into the speaker (The registry is configured in 31-bgp.xml).

To add BGP Peer configuration into the registry, it is necessary to configure regular BGP peer just like in example in 41-bgp-example.xml. Notice that the BGP peer depends on the same *bgp-peer-registry as bgp-speaker*:

```
<module>
    <type
 xmlns:prefix="urn:opendaylight:params:xml:ns:yang:controller:bgp:rib:impl">
        prefix:bgp-peer
    </type>
    <name>example-bgp-peer</name>
    <host>192.0.2.1</host>
    ...
    <peer-registry>
        <type
 xmlns:prefix="urn:opendaylight:params:xml:ns:yang:controller:bgp:rib:impl">
            prefix:bgp-peer-registry
        </type>
        <name>global-bgp-peer-registry</name>
    </peer-registry>
    ...
</module>
```

BGP peer registers itself into the registry, which allows incoming BGP connections handled by the *bgp-speaker*. (Config attribute *peer-registry* is optional for now to preserve backwards compatibility). With this configuration, the connection to 192.0.2.1 is initiated by ODL but will also be accepted from 192.0.2.1. In case both connections are being established, only one of them will be preserved and the other will be dropped. The connection initiated from device with lower bgp id will be dropped by the registry.

There is a way to configure the peer only for incoming connections (The connection will not be initiated by the ODL, ODL will only wait for incoming connection from the peer. The peer is identified by its IP address). To configure peer only for incoming connection add attribute *initiate-connection* to peer configuration:

```
<module>
    <type
 xmlns:prefix="urn:opendaylight:params:xml:ns:yang:controller:bgp:rib:impl">
        prefix:bgp-peer
    </type>
    <name>example-bgp-peer</name>
    <host>192.0.2.1</host>                          // IP address or hostname
 of the speaker
```

```
    <holdtimer>180</holdtimer>
    <initiate-connection>false</initiate-connection>  // Connection will not
 be initiated by ODL
    ...
</module>
```

The attribute initiate-connection is optional with the default value set to **true**.

Application peer configuration

Application peer is a special type of BGP peer. It has own BGP RIB. This RIB can be populated through RESTCONF. If ODL is set as BGP speaker, the changes are sent to other BGP clients as well. To properly configure application peer, add following lines to 41-bgp-example.xml and make appropriate changes.

```
<module>
 <type
 xmlns:prefix="urn:opendaylight:params:xml:ns:yang:controller:bgp:rib:impl">
  prefix:bgp-application-peer
 </type>
 <name>example-bgp-peer-app</name>
 <bgp-id>10.1.9.9</bgp-id> <!-- Your local BGP-ID that will be used in BGP
 Best Path Selection algorithm -->
 <target-rib>
  <type
 xmlns:prefix="urn:opendaylight:params:xml:ns:yang:controller:bgp:rib:impl">
    prefix:rib-instance
  </type>
  <name>example-bgp-rib</name> <!-- RIB where the changes from application RIB
 should be propagated -->
 </target-rib>
 <application-rib-id>example-app-rib</application-rib-id>  <!-- Your
 application RIB identifier -->
 <data-broker>
  <type
 xmlns:binding="urn:opendaylight:params:xml:ns:yang:controller:md:sal:binding">
    binding:binding-async-data-broker
  </type>
  <name>binding-data-broker</name>
 </data-broker>
</module>
```

PCEP

OpenDaylight is pre-configured with baseline PCEP configuration. The default shipped configuration will start a PCE server on port 4189.

32-pcep.xml - basic PCEP configuration, including session parameters 39-pcep-provider.xml - configuration for PCEP provider

Configure draft versions

There are already two extensions for PCEP: draft-ietf-pce-stateful-pce - in versions 02 and 07 draft-ietf-pce-pce-initiated-lsp - versions crabbe-initiated-00 and ietf-initiated-00.

Note

It is important to load the extensions with compatible versions because they extend each other. In this case crabbe-initiated-00 is compatible with stateful-02 and ietf-initiated-00 is compatible with stateful-07. Default configuration is to use newest versions of the drafts.

Complete the following steps in order to get stateful02 PCEP connection running and synchronized.

To use older version: . Switch commented code to ignore stateful-7 and ietf-initiated-00 versions in 32-pcep.xml:

```
 <!-- This block is draft-ietf-pce-stateful-pce-07 + draft-ietf-pce-
inititated-pce-00 -->
 <!--extension>
  <type>pcepspi:extension</type>
  <name>pcep-parser-ietf-stateful07</name>
 </extension>
 <extension>
  <type>pcepspi:extension</type>
  <name>pcep-parser-ietf-initiated00</name>
 </extension-->
 <!-- This block is draft-ietf-pce-stateful-pce-02 + draft-crabbe-pce-
inititated-pce-00 -->
<extension>
 <type
 xmlns:pcepspi="urn:opendaylight:params:xml:ns:yang:controller:pcep:spi">
   pcepspi:extension
 </type>
 <name>pcep-parser-ietf-stateful02</name>
</extension>
<extension>
 <type
 xmlns:pcepspi="urn:opendaylight:params:xml:ns:yang:controller:pcep:spi">
  pcepspi:extension
 </type>
 <name>pcep-parser-crabbe-initiated00</name>
</extension>
```

1. In the same file, make sure the proposal matches your chosen draft version. Change *stateful07-proposal* to *stateful02-proposal*:

```
<pcep-session-proposal-factory>
    <type>pcep:pcep-session-proposal-factory</type>
    <name>stateful02-proposal</name>
</pcep-session-proposal-factory>
```

1. In 39-pcep-provider.xml, stateful-plugin also needs to match. Change *stateful07* to *stateful02*:

```
<stateful-plugin>
    <type>prefix:pcep-topology-stateful</type>
    <name>stateful02</name>
</stateful-plugin>
```

Configure PCEP segment routing

draft-sivabalan-pce-segment-routing-02 PCEP extension for Segment Routing

PCEP Segment Routing initial configuration: 33-pcep-segment-routing.xml

- To use Segment Routing uncomment two commented blocks
- Activate parsers/serializes extension:
  - Create *pcep-parser-segment-routing02* instance
  - Reconfigure (inject into list of extensions) global-pcep-extensions

```
<module>
    <type
 xmlns:prefix="urn:opendaylight:params:xml:ns:yang:controller:pcep:sr02:cfg">
        prefix:pcep-parser-segment-routing02
    </type>
    <name>pcep-parser-segment-routing02</name>
</module>
<module>
    <type
 xmlns:prefix="urn:opendaylight:params:xml:ns:yang:controller:pcep:spi">
        prefix:pcep-extensions-impl
    </type>
    <name>global-pcep-extensions</name>
    <extension>
        <type
 xmlns:pcepspi="urn:opendaylight:params:xml:ns:yang:controller:pcep:spi">
            pcepspi:extension
        </type>
        <name>pcep-parser-segment-routing02</name>
    </extension>
</module>
.
.
.
<services xmlns="urn:opendaylight:params:xml:ns:yang:controller:config">
    <service>
        <type
 xmlns:pcepspi="urn:opendaylight:params:xml:ns:yang:controller:pcep:spi">
            pcepspi:extension
        </type>
        <instance>
            <name>pcep-parser-segment-routing02</name>
            <provider>/config/modules/module[name='pcep-parser-segment-
routing02']/instance[name='pcep-parser-segment-routing02']</provider>
        </instance>
    </service>
</services>
```

- Advertise Segment Routing capability in Open Message:
  - Instantiate pcep-session-proposal-factory-sr02
  - Reconfigure *global-pcep-dispatcher*

```
<module>
```

```
    <type
 xmlns:prefix="urn:opendaylight:params:xml:ns:yang:controller:pcep:sr02:cfg">
        prefix:pcep-session-proposal-factory-sr02
    </type>
    <name>pcep-session-proposal-factory-sr02</name>
</module>
<module>
    <type
 xmlns:prefix="urn:opendaylight:params:xml:ns:yang:controller:pcep:impl">
        prefix:pcep-dispatcher-impl
    </type>
    <name>global-pcep-dispatcher</name>
    <pcep-session-proposal-factory>
        <type
 xmlns:pcep="urn:opendaylight:params:xml:ns:yang:controller:pcep">
            pcep:pcep-session-proposal-factory
        </type>
        <name>pcep-session-proposal-factory-sr02</name>
    </pcep-session-proposal-factory>
</module>
.
.
.
<services xmlns="urn:opendaylight:params:xml:ns:yang:controller:config">
    <service>
        <type
 xmlns:pcep="urn:opendaylight:params:xml:ns:yang:controller:pcep">
            pcep:pcep-session-proposal-factory
        </type>
        <instance>
            <name>pcep-session-proposal-factory-sr02</name>
            <provider>/config/modules/module[name='pcep-session-proposal-
factory-sr02']/instance[name='pcep-session-proposal-factory-sr02']</provider>
        </instance>
    </service>
</services>
```

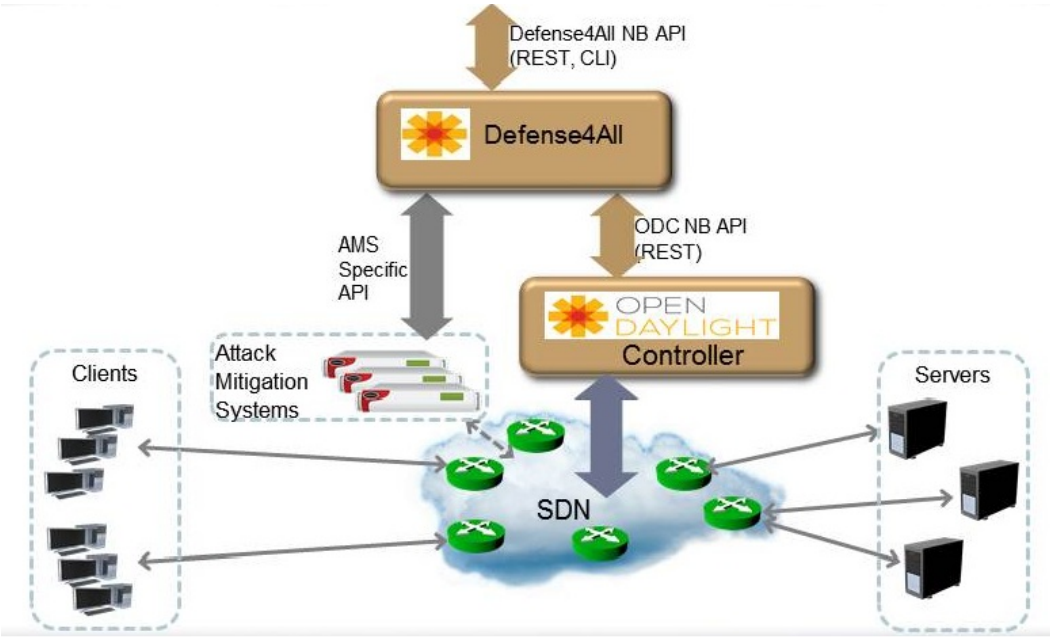
6. Defense4All

Table of Contents

Defense4All Overview

Defense4All is an SDN application for detecting and mitigating DDoS attacks. The figure below depicts the positioning of Defense4All in OpenDaylight environment.

Figure 6.1. Defense4All Overview



The application communicates with OpenDaylight Controller through the ODC north-bound REST API.

Through the REST API Defense4All performs the following tasks:

1. Monitoring behavior of protected traffic - the application sets flow entries in selected network locations to read traffic statistics for each of the PNs (aggregating statistics collected for a given PN from multiple locations).

2. Diverting attacked traffic to selected AMSs – the application set flow entries in selected network locations to divert traffic to selected AMSs. When an attack is over the application removes these flow entries, thus returning to normal operation and traffic monitoring.

Defense4All can optionally communicate with the defined AMSs. For example: To dynamically configure them, monitor them or collect and act upon attack statistics from

the AMSs. The API to AMS is not standardized, and in any case beyond the scope of the OpenDaylight work. Defense4All contains a reference implementation pluggable driver to communicate with Radware's DefensePro AMS.

The application presents its north-bound REST and CLI APIs to allow its manager to:

Control and configure the application (runtime parameters, ODC connectivity, AMSs in domain, PNs, and so on.). Obtain reporting data – operational or security, current or historical, unified from Defense4All and other sources such as, ODC and AMSs). Defense4All provides unified management, reporting and monitoring.

Management - Important part of Defense4All operation is to allow users simple "one touch" and abstracted provisioning of security services, for both detection and mitigation operations. The user needs to only specify simple security attributes. **Reporting and monitoring operations** - Important part of security services is a combination of (near) real-time logs for monitoring as well as historical logs for reporting. Defense4All provides a unified interface for both purposes. The monitoring information is based on various events collected from Defense4All, AMSs and ODC, allowing rich and correlated view on events. Logged event records can be operational or security related. The former includes failures and errors and informational logs. The latter includes detections, attacks and attack mitigation lifecycles, traffic diversion information and periodic traffic averages. All logs are persistent (stable storage and replication).

Defense4All User Interface

This section describes how to configure the Defense4All Framework environment.

Configuring the FrameWork Environment

To set Defense4All configuration parameters:

1. From an Internet browser, go to http://<ip address>:8086/controlapps, where *<ip address>* is the address for the host that is running Defense4All.
2. From the FrameWork Setup pane, select **Framework** > **Setup**.
3. Set the **Framework Control Network Address** to the IP address Defense4All uses to access the control network.
4. To the right of the SDN Controllers label, click **Add**.
5. In the Add SDN Controller pane, set the following parameters:

Parameter	Description
Hostname	Name of the SDN Controller. This is the SDN Controller that supports OpenFlow network programming (OFC stands for OpenFlow Controller). OpenDaylight Controller provides this flavor both for OpenFlow enabled network devices and other network devices with adequate plug-ins in the PFC.
IP address	IP address of the SDN Controller.
Port	Port number of the SDN Controller.
Statistics Polling Interval	The frequency that the SDN Controller polls for statistics.
Username	Username to log into the SDN Controller.

Parameter	Description
Password	Password to log into the SDN Controller.
Confirm Password	Confirmation of the password of the SDN Controller.

1. Click Submit.

Note

The SDN controller cannot be changed or removed. Only one (1) SDN controller can be configured. To change the SDN controller, you must reset Defense4All to factory settings. . In the FrameWork Setup pane, to the right of the Attack Mitigation Systems (AMSs) label, click Add. . In the Add Attack Mitigation System (AMS) pane, set the following parameters:

Parameter	Description
Name	AMS descriptive name.
Brand	Select the AMS brand from the drop-down list. Values: Radware DefensePro, Other Default: Radware **Note:** The Radware DefensePro device can be removed only when there are no active mitigations (traffic redirections to it).
Version	AMS version. **Note:** This parameter is only applicable to Radware DefensePro.
IP Address	AMS IP address. **Note:** This parameter is only applicable to Radware DefensePro.
Port	AMS port number. **Note:** This parameter is only applicable to Radware DefensePro.
Username	AMS username. **Note:** This parameter is only applicable to Radware DefensePro.
Password	Password to log into the AMS. **Note:** This parameter is only applicable to Radware DefensePro.
Confirm Password	Confirmation of the password of the AMS. **Note:** This parameter is only applicable to Radware DefensePro.
Health Check	Interval Time in seconds. **Note:** This parameter is only applicable to Radware DefensePro. Default: 60 seconds

Note

Only relevant for DefensePro. Layer 2 Broadcast Destination MAC Address, Multicast Destination MAC Address, Unrecognized L2 Format, and TTL Less

Than or Equal to 1 blocking must be configured to avoid Layer 2 loops. For more information, refer to the discussion on Packet Anomaly protection in the DefensePro User Guide.

1. Click **Submit**.
2. In the FrameWork Setup pane, to the right of the **Net Nodes** label, click **Add**.
3. In the Add Net Node pane, set the following parameters:

Parameter	Description
Name	NetNode descriptive name.
ID	NetNode ID.
Type (read-only)	Default: Openflow
SDN Node Mode (read-only)	Default: sdnenablednative.
Health Check Interval (read- only)	Default: 60 seconds

4. To the right of the **Protected Links** label, click Add.
5. In the Add Protected Link pane, set the following parameters:

Parameter	Description
Incoming Traffic Port	The incoming traffic port number.
Outgoing Traffic Port	The outgoing traffic port number.

6. Click **OK**.
7. To the right of the AMS Connections label, click **Add**.
8. In the Add AMS Connection pane, set the following parameters:

Parameter	Description
Name	AMS connection descriptive name.
AMS Name	AMS connection name.
NetNode North Port	NetNode NothPort.
NetNode South Port	NetNode South Port.
AMS North Port	AMS North Port.
AMS South Port	AMS South Port.

9. Click **OK**.
10. In the Add Net Node pane, click **Submit**.

FrameWork Maintenance

This section describes how to run maintenance operations on Defense4All

- **Reset to Factory Settings** — If you want to reset Defense4All to its factory settings, at the bottom of the FrameWork Setup pane, click Reset to Factory Settings.
- **Restart Framework** — To manually restart Defense4All, at the bottom of the FrameWork Setup pane, click Restart FrameWork.

FrameWork Reports

You can generate reports containing syslog messages that have been saved over a period of time.

To generate FrameWork reports:

1. From an Internet browser enter the IP address for the host that is running Defense4All.
2. In the FrameWork Reports pane, select **Framework > Report**.
3. In the FrameWork Report pane, select one of the tabs:
   a. Query by Time Period
      - In the **From** and **To** fields, select the appropriate dates to define the range of the query.
      - Select the **Event Types** you want included in the report.
      - Click **Run Query**. The results display at the bottom of the pane.
      - Enter a file path in the **Filename** filed, and click **Export Query to File** to save the query to a file.
   b. Query by Last Number of Rows
      - In the **Number of Rows** field, enter the last number of rows in the database you want displayed in your report.
      - Select the **Event Types** you want included in the report.
      - Click **Run Query**. The results display at the bottom of the pane. You cannot save this query to a file
   c. Cleanup
      - In the **Delete events older than** field, enter a number of days. Events older than this number of days are deleted.
      - Click **Submit**. The results display at the bottom of the pane. You cannot save this query to a file.

Configuring Defense4All Protected Objects (POs)

This section describes how to configure Defense4All protected objects (POs).

To set up Defense4All protected objects (POs):

1. From an Internet browser, enter the IP address for the host that is running Defense4All.
2. From the Defense4All Setup pane, select **Defense4All** > **Setup**.
3. To the right of the **Protected Objects (POs)** label, click **Add**.

4. In the Add Protected Object (PO) pane, set the following parameters:

Parameter	Description
Name	Name of the PO. Valid characters: A–Z, a–z, 0-9, _ **NOTE:** A PO cannot be removed when under attack.
IP Address	IP address and net mask of the PO.

1. Click Submit.

Defense4All Reports

You can generate reports containing syslog messages that have been saved over a period of time.

To generate Defense4All reports:

1. From an Internet browser enter the IP address for the host that is running Defense4All.
2. In the Defense4All Reports pane, select **Defense4All** > **Report**.
3. In the Defense4All Reports pane, select one of the tabs:

-Query by Time Period

- In the **From** and **To** fields, select the appropriate dates to define the range of the query.
- Select the **Event Types** you want included in the report.
- Click **Run Queryv**. The results display at the bottom of the pane.
- To save the query to a file, enter a file path in the **Filename** filed, and click **Export Query** to File.

-Query by Last Number of Rows

- In the **Number of Rows** field, enter the last number of rows in the database you want displayed in your report.
- Select the **Event Types** you want included in the report.
- Click **Run Query**. The results display at the bottom of the pane. You cannot save this query to a file.

-Cleanup

- In the **Delete events older** than field, enter a number of days. Events older than this number of days are deleted.
- Click **Submit**. The results display at the bottom of the pane. You cannot save this query to a file.

7. Group-Based Policy

Table of Contents

The Group-Based Policy implementation for Helium is a Proof of Concept. This Proof of Concept implementation includes one example of a group-based policy renderer, based on Open vSwitch, OpenFlow, and OVSDB. Users can create policies and endpoints using the RESTCONF northbound API.

Architecture and Model

The Group-Based Policy architecture and model are described in the OpenDaylight Developer's Guide.

Tutorial

This section will walk you through setting up a simple demo of the OpenFlow overlay renderer using mininet. This will simulate a scenario with two VM hosts connected over a VXLAN tunnel.

Prepare the Environment

Start with two running Ubuntu 14.04 systems, which can be either VMs or physical machines. You'll need a newer version of openvswitch than exists in Ubuntu 14.04, but you only need the user space components so this is easy. We'll start by installing OVS 2.1.2 or later.

Log into one of your Ubuntu systems, and run:

```
OVS_VERSION=2.1.2
sudo apt-get install build-essential fakeroot debhelper libssl-dev
wget http://openvswitch.org/releases/openvswitch-${OVS_VERSION}.tar.gz
tar -xzf openvswitch-${OVS_VERSION}.tar.gz
cd openvswitch-${OVS_VERSION}
DEB_BUILD_OPTIONS='parallel=8 nocheck' fakeroot debian/rules binary
cd ..
sudo dpkg -i openvswitch-common_${OVS_VERSION}-1_amd64.deb openvswitch-
switch_${OVS_VERSION}-1_amd64.deb
sudo apt-get install mininet
```

Now, either run the same commands on the other system, or just copy the openvswitch-common and openvswitch-switch deb files over and install them, plus install mininet from apt.

Configuring the Test

The test script is found in the source tree under `util/testOfOverlay`. Copy the `.py` files from this directory to each of your test systems. Open `config.py` in an editor. You can play with this file later, but for now, just find the section that reads:

```
switches = [{'name': 's1',
             'tunnelIp': '10.160.9.20',
             'dpid': '1'},
            {'name': 's2',
             'tunnelIp': '10.160.9.21',
             'dpid': '2'}]
```

Change the `tunnelIp` items to be the IP addresses of each of your test systems. The IP address of host 1 should be assigned to s1 and similarly for host 2 and s2.

Running the Test

Now, run the controller. You can run it on one of your test systems or on a third system.

On test host 1, cd to the directory containing the `testOfOverlay` script and run:

```
CONTROLLER=10.160.31.238
sudo ./testOfOverlay.py --local s1 --controller ${CONTROLLER}
```

You'll need to replace the `CONTROLLER` address with the IP address of the system where you ran your controller. This will run mininet and set up the hosts that are configured as attached to s1. When you're finished running this, you'll be at a mininet prompt, but you won't be able to do anything because the policy is not set up.

The output will look like:

```
$ sudo ./testOfOverlay.py --local s1 --controller 10.160.31.238
*** Configuring hosts
h35_2 h35_3 h36_2 h36_3
*** Starting controller
*** Starting 1 switches
s1
POST http://10.160.31.238:8080/restconf/operations/endpoint:register-endpoint
{
    "input": {
        "endpoint-group": "1eaf9a67-a171-42a8-9282-71cf702f61dd",
        "l2-context": "70aeb9ea-4ca1-4fb9-9780-22b04b84a0d6",
        "l3-address": [
            {
                "ip-address": "10.0.35.2",
                "l3-context": "f2311f52-890f-4095-8b85-485ec8b92b3c"
            }
        ],
        "mac-address": "00:00:00:00:35:02",
        "ofoverlay:node-connector-id": "openflow:1:1",
        "ofoverlay:node-id": "openflow:1",
        "tenant": "f5c7d344-d1c7-4208-8531-2c2693657e12"
    }
}

POST http://10.160.31.238:8080/restconf/operations/endpoint:register-endpoint
```

```
{
    "input": {
        "endpoint-group": "1eaf9a67-a171-42a8-9282-71cf702f61dd",
        "l2-context": "70aeb9ea-4ca1-4fb9-9780-22b04b84a0d6",
        "l3-address": [
            {
                "ip-address": "10.0.35.3",
                "l3-context": "f2311f52-890f-4095-8b85-485ec8b92b3c"
            }
        ],
        "mac-address": "00:00:00:00:35:03",
        "ofoverlay:node-connector-id": "openflow:1:2",
        "ofoverlay:node-id": "openflow:1",
        "tenant": "f5c7d344-d1c7-4208-8531-2c2693657e12"
    }
}

POST http://10.160.31.238:8080/restconf/operations/endpoint:register-endpoint
{
    "input": {
        "endpoint-group": "e593f05d-96be-47ad-acd5-ba81465680d5",
        "l2-context": "70aeb9ea-4ca1-4fb9-9780-22b04b84a0d6",
        "l3-address": [
            {
                "ip-address": "10.0.36.2",
                "l3-context": "f2311f52-890f-4095-8b85-485ec8b92b3c"
            }
        ],
        "mac-address": "00:00:00:00:36:02",
        "ofoverlay:node-connector-id": "openflow:1:3",
        "ofoverlay:node-id": "openflow:1",
        "tenant": "f5c7d344-d1c7-4208-8531-2c2693657e12"
    }
}

POST http://10.160.31.238:8080/restconf/operations/endpoint:register-endpoint
{
    "input": {
        "endpoint-group": "e593f05d-96be-47ad-acd5-ba81465680d5",
        "l2-context": "70aeb9ea-4ca1-4fb9-9780-22b04b84a0d6",
        "l3-address": [
            {
                "ip-address": "10.0.36.3",
                "l3-context": "f2311f52-890f-4095-8b85-485ec8b92b3c"
            }
        ],
        "mac-address": "00:00:00:00:36:03",
        "ofoverlay:node-connector-id": "openflow:1:4",
        "ofoverlay:node-id": "openflow:1",
        "tenant": "f5c7d344-d1c7-4208-8531-2c2693657e12"
    }
}

*** Starting CLI:
mininet>
```

On test host 2, you'll do the same but run instead:

```
CONTROLLER=10.160.31.238
sudo ./testOfOverlay.py --local s2 --controller ${CONTROLLER} --policy
```

This will run mininet on the other system, and also install all the policy required to enable the connectivity.

The output will look like:

```
$ sudo ./testOfOverlay.py --local s2 --controller ${CONTROLLER} --policy
*** Configuring hosts
h35_4 h35_5 h36_4 h36_5
*** Starting controller
*** Starting 1 switches
s2
PUT http://10.160.31.238:8080/restconf/config/opendaylight-inventory:nodes
{
    "opendaylight-inventory:nodes": {
        "node": [
            {
                "id": "openflow:1",
                "ofoverlay:tunnel-ip": "10.160.9.20"
            },
            {
                "id": "openflow:2",
                "ofoverlay:tunnel-ip": "10.160.9.21"
            }
        ]
    }
}

PUT http://10.160.31.238:8080/restconf/config/policy:tenants
{
    "policy:tenants": {
        "tenant": [
            {
                "contract": [
                    {
                        "clause": [
                            {
                                "name": "allow-http-clause",
                                "subject-refs": [
                                    "allow-http-subject",
                                    "allow-icmp-subject"
                                ]
                            }
                        ],
                        "id": "22282cca-9a13-4d0c-a67e-a933ebb0b0ae",
                        "subject": [
                            {
                                "name": "allow-http-subject",
                                "rule": [
                                    {
                                        "classifier-ref": [
                                            {
                                                "direction": "in",
                                                "name": "http-dest"
                                            },
                                            {
                                                "direction": "out",
                                                "name": "http-src"
                                            }
                                        ],
                                        "name": "allow-http-rule"
```

```
                                    }
                                ]
                            },
                            {
                                "name": "allow-icmp-subject",
                                "rule": [
                                    {
                                        "classifier-ref": [
                                            {
                                                "name": "icmp"
                                            }
                                        ],
                                        "name": "allow-icmp-rule"
                                    }
                                ]
                            }
                        ]
                    }
                ],
                "endpoint-group": [
                    {
                        "consumer-named-selector": [
                            {
                                "contract": [
                                    "22282cca-9a13-4d0c-a67e-a933ebb0b0ae"
                                ],
                                "name": "e593f05d-96be-47ad-acd5-
ba81465680d5-1eaf9a67-a171-42a8-9282-71cf702f61dd-22282cca-9a13-4d0c-a67e-
a933ebb0b0ae"
                            }
                        ],
                        "id": "1eaf9a67-a171-42a8-9282-71cf702f61dd",
                        "network-domain": "77284c12-a569-4585-b244-
af9b078acfe4",
                        "provider-named-selector": []
                    },
                    {
                        "consumer-named-selector": [],
                        "id": "e593f05d-96be-47ad-acd5-ba81465680d5",
                        "network-domain": "472ab051-554e-45be-
a133-281f0a53412a",
                        "provider-named-selector": [
                            {
                                "contract": [
                                    "22282cca-9a13-4d0c-a67e-a933ebb0b0ae"
                                ],
                                "name": "e593f05d-96be-47ad-acd5-
ba81465680d5-1eaf9a67-a171-42a8-9282-71cf702f61dd-22282cca-9a13-4d0c-a67e-
a933ebb0b0ae"
                            }
                        ]
                    }
                ],
                "id": "f5c7d344-d1c7-4208-8531-2c2693657e12",
                "l2-bridge-domain": [
                    {
                        "id": "70aeb9ea-4ca1-4fb9-9780-22b04b84a0d6",
                        "parent": "f2311f52-890f-4095-8b85-485ec8b92b3c"
                    }
                ],
```

```
                "l2-flood-domain": [
                    {
                        "id": "34cc1dd1-2c8c-4e61-a177-588b2d4133b4",
                        "parent": "70aeb9ea-4ca1-4fb9-9780-22b04b84a0d6"
                    },
                    {
                        "id": "6e669acf-2fd9-48ea-a9b0-cd98d933a6b8",
                        "parent": "70aeb9ea-4ca1-4fb9-9780-22b04b84a0d6"
                    }
                ],
                "l3-context": [
                    {
                        "id": "f2311f52-890f-4095-8b85-485ec8b92b3c"
                    }
                ],
                "subject-feature-instances": {
                    "classifier-instance": [
                        {
                            "classifier-definition-id": "4250ab32-e8b8-445a-
aebb-e1bd2cdd291f",
                            "name": "http-dest",
                            "parameter-value": [
                                {
                                    "name": "type",
                                    "string-value": "TCP"
                                },
                                {
                                    "int-value": "80",
                                    "name": "destport"
                                }
                            ]
                        },
                        {
                            "classifier-definition-id": "4250ab32-e8b8-445a-
aebb-e1bd2cdd291f",
                            "name": "http-src",
                            "parameter-value": [
                                {
                                    "name": "type",
                                    "string-value": "TCP"
                                },
                                {
                                    "int-value": "80",
                                    "name": "sourceport"
                                }
                            ]
                        },
                        {
                            "classifier-definition-id": "79c6fdb2-1e1a-4832-
af57-c65baf5c2335",
                            "name": "icmp",
                            "parameter-value": [
                                {
                                    "int-value": "1",
                                    "name": "proto"
                                }
                            ]
                        }
                    ]
                },
```

```
                "subnet": [
                    {
                        "id": "77284c12-a569-4585-b244-af9b078acfe4",
                        "ip-prefix": "10.0.35.1/24",
                        "parent": "34cc1dd1-2c8c-4e61-a177-588b2d4133b4",
                        "virtual-router-ip": "10.0.35.1"
                    },
                    {
                        "id": "472ab051-554e-45be-a133-281f0a53412a",
                        "ip-prefix": "10.0.36.1/24",
                        "parent": "6e669acf-2fd9-48ea-a9b0-cd98d933a6b8",
                        "virtual-router-ip": "10.0.36.1"
                    }
                ]
            }
        ]
    }
}

POST http://10.160.31.238:8080/restconf/operations/endpoint:register-endpoint
{
    "input": {
        "endpoint-group": "1eaf9a67-a171-42a8-9282-71cf702f61dd",
        "l2-context": "70aeb9ea-4ca1-4fb9-9780-22b04b84a0d6",
        "l3-address": [
            {
                "ip-address": "10.0.35.4",
                "l3-context": "f2311f52-890f-4095-8b85-485ec8b92b3c"
            }
        ],
        "mac-address": "00:00:00:00:35:04",
        "ofoverlay:node-connector-id": "openflow:2:1",
        "ofoverlay:node-id": "openflow:2",
        "tenant": "f5c7d344-d1c7-4208-8531-2c2693657e12"
    }
}

POST http://10.160.31.238:8080/restconf/operations/endpoint:register-endpoint
{
    "input": {
        "endpoint-group": "1eaf9a67-a171-42a8-9282-71cf702f61dd",
        "l2-context": "70aeb9ea-4ca1-4fb9-9780-22b04b84a0d6",
        "l3-address": [
            {
                "ip-address": "10.0.35.5",
                "l3-context": "f2311f52-890f-4095-8b85-485ec8b92b3c"
            }
        ],
        "mac-address": "00:00:00:00:35:05",
        "ofoverlay:node-connector-id": "openflow:2:2",
        "ofoverlay:node-id": "openflow:2",
        "tenant": "f5c7d344-d1c7-4208-8531-2c2693657e12"
    }
}

POST http://10.160.31.238:8080/restconf/operations/endpoint:register-endpoint
{
    "input": {
        "endpoint-group": "e593f05d-96be-47ad-acd5-ba81465680d5",
        "l2-context": "70aeb9ea-4ca1-4fb9-9780-22b04b84a0d6",
```

```
        "l3-address": [
            {
                "ip-address": "10.0.36.4",
                "l3-context": "f2311f52-890f-4095-8b85-485ec8b92b3c"
            }
        ],
        "mac-address": "00:00:00:00:36:04",
        "ofoverlay:node-connector-id": "openflow:2:3",
        "ofoverlay:node-id": "openflow:2",
        "tenant": "f5c7d344-d1c7-4208-8531-2c2693657e12"
    }
}

POST http://10.160.31.238:8080/restconf/operations/endpoint:register-endpoint
{
    "input": {
        "endpoint-group": "e593f05d-96be-47ad-acd5-ba81465680d5",
        "l2-context": "70aeb9ea-4ca1-4fb9-9780-22b04b84a0d6",
        "l3-address": [
            {
                "ip-address": "10.0.36.5",
                "l3-context": "f2311f52-890f-4095-8b85-485ec8b92b3c"
            }
        ],
        "mac-address": "00:00:00:00:36:05",
        "ofoverlay:node-connector-id": "openflow:2:4",
        "ofoverlay:node-id": "openflow:2",
        "tenant": "f5c7d344-d1c7-4208-8531-2c2693657e12"
    }
}

*** Starting CLI:
mininet>
```

Verifying

In the default test, we have a total of 2 hosts on each switch in each of 2 endpoint groups, for a total of eight hosts. The endpoints are in two different subnets, so communicating across the two endpoint groups requires routing. There is a contract set up that allows HTTP from EG1 to EG2, and ICMP in both directions between EG1 and EG2.

ICMP

We expect ICMP to work between all pairs of hosts. First, on host one, run pingall as follows:

```
mininet> pingall
*** Ping: testing ping reachability
h35_2 -> h35_3 h36_2 h36_3
h35_3 -> h35_2 h36_2 h36_3
h36_2 -> h35_2 h35_3 h36_3
h36_3 -> h35_2 h35_3 h36_2
*** Results: 0% dropped (12/12 received)
```

and the same on host 2:

```
mininet> pingall
*** Ping: testing ping reachability
```

```
h35_4 -> h35_5 h36_4 h36_5
h35_5 -> h35_4 h36_4 h36_5
h36_4 -> h35_4 h35_5 h36_5
h36_5 -> h35_4 h35_5 h36_4
```

The hosts h35_[n] are in EG1, in the subnet 10.0.35.1/24. Hosts h36_[n] are in EG2, in the subnet 10.0.36.1/24. These two tests therefore shows broadcast within the flood domain working to enable ARP, bridging within the endpoint group, and the functioning of the virtual router which is routing traffic between the two subnets. It also shows the ICMP policy allowing the ping between the two groups.

Now we can test connectivity over the tunnel:

```
mininet> h35_2 ping -c1 10.0.35.4
PING 10.0.35.4 (10.0.35.4) 56(84) bytes of data.
64 bytes from 10.0.35.4: icmp_seq=1 ttl=64 time=1.78 ms

--- 10.0.35.4 ping statistics ---
1 packets transmitted, 1 received, 0% packet loss, time 0ms
rtt min/avg/max/mdev = 1.786/1.786/1.786/0.000 ms
mininet> h35_2 ping -c1 10.0.35.5
PING 10.0.35.5 (10.0.35.5) 56(84) bytes of data.
64 bytes from 10.0.35.5: icmp_seq=1 ttl=64 time=2.59 ms

--- 10.0.35.5 ping statistics ---
1 packets transmitted, 1 received, 0% packet loss, time 0ms
rtt min/avg/max/mdev = 2.597/2.597/2.597/0.000 ms
mininet> h35_2 ping -c1 10.0.36.4
PING 10.0.36.4 (10.0.36.4) 56(84) bytes of data.
64 bytes from 10.0.36.4: icmp_seq=1 ttl=62 time=2.64 ms

--- 10.0.36.4 ping statistics ---
1 packets transmitted, 1 received, 0% packet loss, time 0ms
rtt min/avg/max/mdev = 2.641/2.641/2.641/0.000 ms
mininet> h35_2 ping -c1 10.0.36.5
PING 10.0.36.5 (10.0.36.5) 56(84) bytes of data.
64 bytes from 10.0.36.5: icmp_seq=1 ttl=62 time=2.93 ms

--- 10.0.36.5 ping statistics ---
1 packets transmitted, 1 received, 0% packet loss, time 0ms
rtt min/avg/max/mdev = 2.936/2.936/2.936/0.000 ms
```

This shows all those same features working transparently across the tunnel to the hosts on the other switch.

HTTP

We expect HTTP to work only when going from EG1 to EG2, and only on port 80. Let's check. First, we'll start a web server on h36_2 by running this on host 1:

```
mininet> h36_2 python -m SimpleHTTPServer 80
```

Note that this will block your prompt until you Ctrl-C it later.

Now on host 2, run:

```
mininet> h35_4 curl http://10.0.36.2
  % Total    % Received % Xferd  Average Speed   Time    Time     Time
 Current
```

```
                                 Dload  Upload   Total   Spent    Left  Speed
100   488  100   488    0     0  72944      0 --:--:-- --:--:-- --:--:-- 97600
<!DOCTYPE html PUBLIC "-//W3C//DTD HTML 3.2 Final//EN"><html>
<title>Directory listing for /</title>
<body>
<h2>Directory listing for /</h2>
<hr>
<ul>
<li><a href="config.py">config.py</a>
<li><a href="config.pyc">config.pyc</a>
<li><a href="mininet_gbp.py">mininet_gbp.py</a>
<li><a href="mininet_gbp.pyc">mininet_gbp.pyc</a>
<li><a href="odl_gbp.py">odl_gbp.py</a>
<li><a href="odl_gbp.pyc">odl_gbp.pyc</a>
<li><a href="testOfOverlay.py">testOfOverlay.py</a>
</ul>
<hr>
</body>
</html>
```

You can see that the host in endpoint group 1 is able to access the server in endpoint group 2.

Let's try the reverse. Ctrl-C the server on host 1 and then run:

```
mininet> h35_2 python -m SimpleHTTPServer 80
```

We can still access the server from h35_4 on host 2, because it's in the same endpoint group:

```
mininet> h35_4 curl http://10.0.35.2
  % Total    % Received % Xferd  Average Speed   Time    Time     Time
 Current
                                 Dload  Upload   Total   Spent    Left  Speed
100   488  100   488    0     0  55625      0 --:--:-- --:--:-- --:--:-- 61000
<!DOCTYPE html PUBLIC "-//W3C//DTD HTML 3.2 Final//EN"><html>
<title>Directory listing for /</title>
<body>
<h2>Directory listing for /</h2>
<hr>
<ul>
<li><a href="config.py">config.py</a>
<li><a href="config.pyc">config.pyc</a>
<li><a href="mininet_gbp.py">mininet_gbp.py</a>
<li><a href="mininet_gbp.pyc">mininet_gbp.pyc</a>
<li><a href="odl_gbp.py">odl_gbp.py</a>
<li><a href="odl_gbp.pyc">odl_gbp.pyc</a>
<li><a href="testOfOverlay.py">testOfOverlay.py</a>
</ul>
<hr>
</body>
</html>
```

But we cannot access it from h36_4 on host 2, because it's in a different endpoint group and our contract allows HTTP only in the other direction:

```
mininet> h36_4 curl http://10.0.35.2 --connect-timeout 3
  % Total    % Received % Xferd  Average Speed   Time    Time     Time
 Current
                                 Dload  Upload   Total   Spent    Left  Speed
```

```
  0     0    0     0    0     0      0      0 --:--:--  0:00:03 --:--:--     0
curl: (28) Connection timed out after 3001 milliseconds
```

Contact Information

Mailing List	groupbasedpolicy-users@lists.opendaylight.org
IRC	freenode.net #opendaylight-group-policy
Repository	https://git.opendaylight.org/gerrit/groupbasedpolicy

8. L2Switch

Table of Contents

The L2Switch project provides Layer2 switch functionality.

Running the L2Switch project

Check out the project using git

```
git clone https://git.opendaylight.org/gerrit/p/l2switch.git
```

The above command will create a directory called "l2switch" with the project.

Run the distribution

To run the base distribution, you can use the following command

```
./distribution/base/target/distributions-l2switch-base-0.1.0-SNAPSHOT-
osgipackage/opendaylight/run.sh
```

If you need additional resources, you can use these command line arguments:

```
-Xms1024m -Xmx2048m -XX:PermSize=512m -XX:MaxPermSize=1024m'
```

To run the karaf distribution, you can use the following command:

```
./distribution/karaf/target/assembly/bin/karaf
```

Create a network using mininet

```
sudo mn --controller=remote,ip=<Controller IP> --topo=linear,3 --switch
 ovsk,protocols=OpenFlow13
sudo mn --controller=remote,ip=127.0.0.1 --topo=linear,3 --switch
 ovsk,protocols=OpenFlow13
```

The above command will create a virtual network consisting of 3 switches. Each switch will connect to the controller located at the specified IP, i.e. 127.0.0.1

```
sudo mn --controller=remote,ip=127.0.0.1 --mac --topo=linear,3 --switch
 ovsk,protocols=OpenFlow13
```

The above command has the "mac" option, which makes it easier to distinguish between Host MAC addresses and Switch MAC addresses.

Generating network traffic using mininet

```
h1 ping h2
```

The above command will cause host1 (h1) to ping host2 (h2)

```
pingall
```

pingall will cause each host to ping every other host.

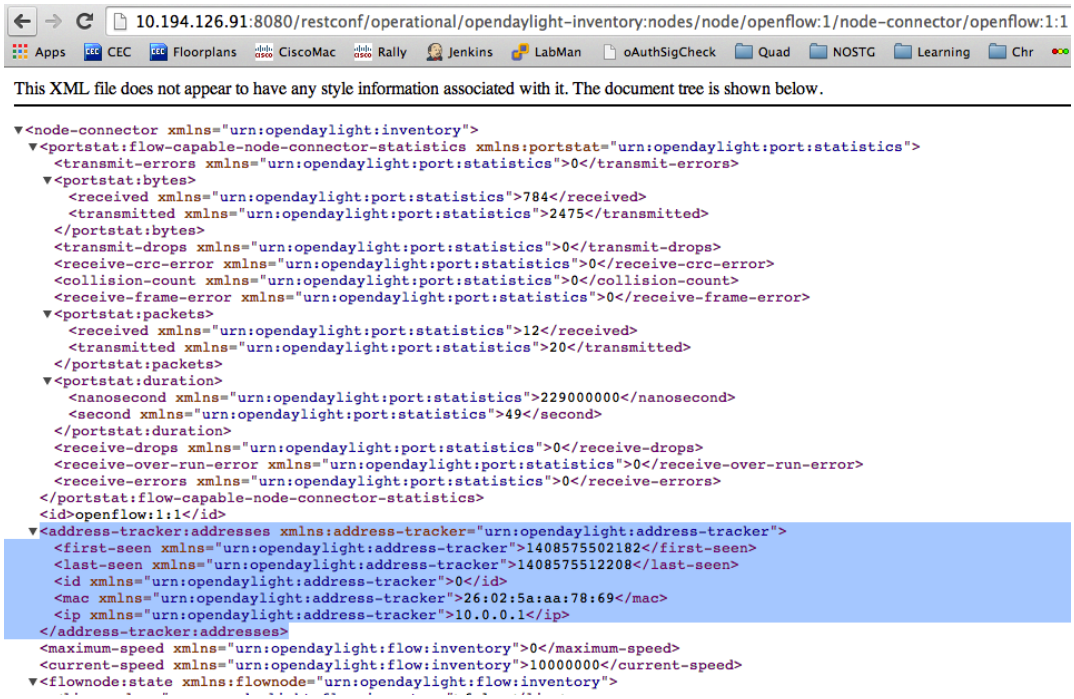
Checking Address Observations

Address Observations are added to the Inventory data tree.

The Address Observations on a Node Connector can be checked through a browser or a REST Client.

```
http://10.194.126.91:8080/restconf/operational/opendaylight-inventory:nodes/
node/openflow:1/node-connector/openflow:1:1
```

Figure 8.1. Address Observations



Checking Hosts

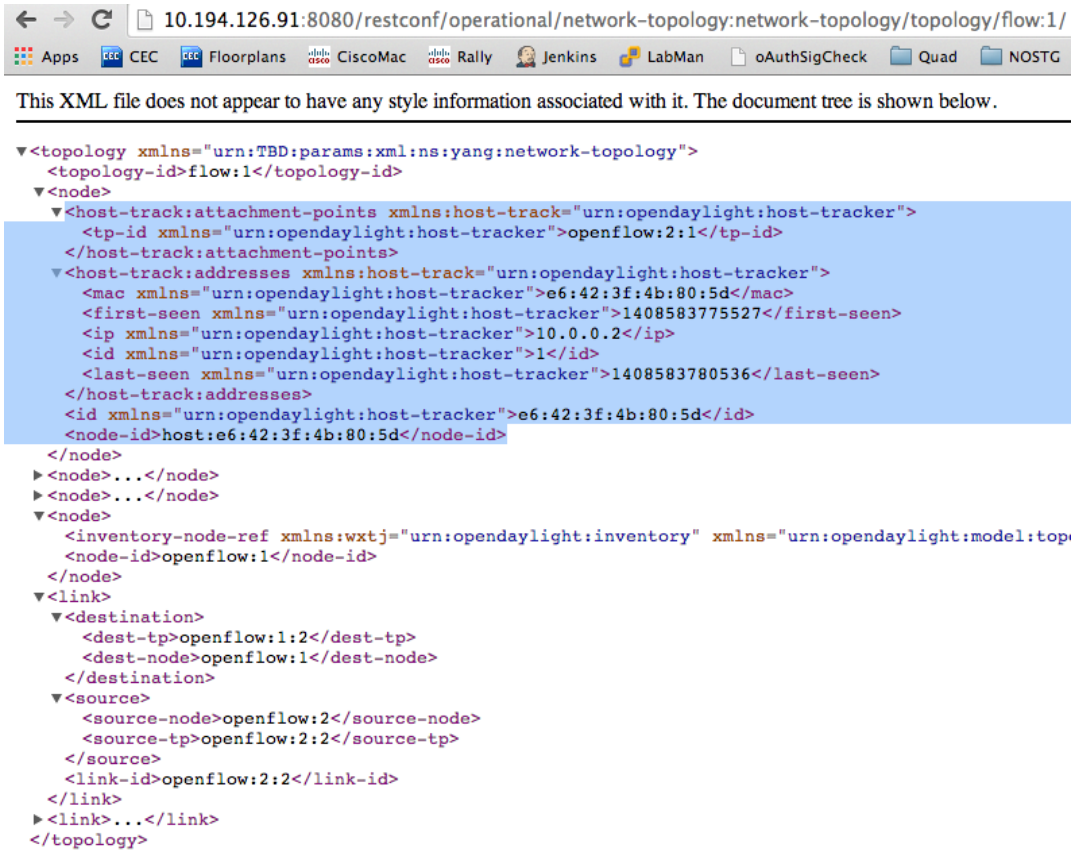
Host information is added to the Topology data tree.

- Host address
- Attachment point (link) to a node/switch

This host information and attachment point information can be checked through a browser or a REST Client.

```
http://10.194.126.91:8080/restconf/operational/network-topology:network-
topology/topology/flow:1/
```

Figure 8.2. Hosts



Checking STP status of each link

STP Status information is added to the Inventory data tree.

- A status of "forwarding" means the link is active and packets are flowing on it.
- A status of "discarding" means the link is inactive and packets are not sent over it.

The STP status of a link can be checked through a browser or a REST Client.

```
http://10.194.126.91:8080/restconf/operational/opendaylight-inventory:nodes/
node/openflow:1/node-connector/openflow:1:2
```

Figure 8.3. STP status

Miscellaneous mininet commands

```
link s1 s2 down
```

This will bring the link between switch1 (s1) and switch2 (s2) down

```
link s1 s2 up
```

This will bring the link between switch1 (s1) and switch2 (s2) up

```
link s1 h1 down
```

This will bring the link between switch1 (s1) and host1 (h1) down

Components of the L2Switch

- Packet Handler
  - Decodes the packets coming to the controller and dispatches them appropriately
- Loop Remover
  - Removes loops in the network
- Arp Handler
  - Handles the decoded ARP packets
- Address Tracker
  - Learns the Addresses (MAC and IP) of entities in the network

- Host Tracker
  - Tracks the locations of hosts in the network
- L2Switch Main
  - Installs flows on each switch based on network traffic

Configuration of L2Switch Components

This section details the configuration settings for the components that can be configured.

The base distribution configuration files are located in distribution/base/target/distributions-l2switch-base-0.1.0-SNAPSHOT-osgipackage/opendaylight/configuration/initial

The karaf distribution configuration files are located in distribution/karaf/target/assembly/etc/opendaylight/karaf

- Loop Remover (52-loopremover.xml)
  - is-install-lldp-flow
    - "true" means a flow that sends all LLDP packets to the controller will be installed on each switch
    - "false" means this flow will not be installed
  - lldp-flow-table-id
    - The LLDP flow will be installed on the specified flow table of each switch
    - This field is only relevant when "is-install-lldp-flow" is set to "true"
  - lldp-flow-priority
    - The LLDP flow will be installed with the specified priority
    - This field is only relevant when "is-install-lldp-flow" is set to "true"
  - lldp-flow-idle-timeout
    - The LLDP flow will timeout (removed from the switch) if the flow doesn't forward a packet for *x* seconds
    - This field is only relevant when "is-install-lldp-flow" is set to "true"
  - lldp-flow-hard-timeout
    - The LLDP flow will timeout (removed from the switch) after *x* seconds, regardless of how many packets it is forwarding
    - This field is only relevant when "is-install-lldp-flow" is set to "true"
  - graph-refresh-delay

        - A graph of the network is maintained and gets updated as network elements go up/down (i.e. links go up/down and switches go up/down)
        - After a network element going up/down, it waits *graph-refresh-delay* seconds before recomputing the graph
        - A higher value has the advantage of doing less graph updates, at the potential cost of losing some packets because the graph didn't update immediately.
        - A lower value has the advantage of handling network topology changes quicker, at the cost of doing more computation.
    - Arp Handler (54-arphandler.xml)
        - is-proactive-flood-mode
            - "true" means that flood flows will be installed on each switch. With this flood flow, each switch will flood a packet that doesn't match any other flows.
                - Advantage: Fewer packets are sent to the controller because those packets are flooded to the network.
                - Disadvantage: A lot of network traffic is generated.
            - "false" means the previously mentioned flood flows will not be installed. Instead an ARP flow will be installed on each switch that sends all ARP packets to the controller.
                - Advantage: Less network traffic is generated.
                - Disadvantage: The controller handles more packets (ARP requests & replies) and the ARP process takes longer than if there were flood flows.
        - flood-flow-table-id
            - The flood flow will be installed on the specified flow table of each switch
            - This field is only relevant when "is-proactive-flood-mode" is set to "true"
        - flood-flow-priority
            - The flood flow will be installed with the specified priority
            - This field is only relevant when "is-proactive-flood-mode" is set to "true"
        - flood-flow-idle-timeout
            - The flood flow will timeout (removed from the switch) if the flow doesn't forward a packet for *x* seconds
            - This field is only relevant when "is-proactive-flood-mode" is set to "true"
        - flood-flow-hard-timeout

        - The flood flow will timeout (removed from the switch) after *x* seconds, regardless of how many packets it is forwarding
        - This field is only relevant when "is-proactive-flood-mode" is set to "true"
    - arp-flow-table-id
        - The ARP flow will be installed on the specified flow table of each switch
        - This field is only relevant when "is-proactive-flood-mode" is set to "false"
    - arp-flow-priority
        - The ARP flow will be installed with the specified priority
        - This field is only relevant when "is-proactive-flood-mode" is set to "false"
    - arp-flow-idle-timeout
        - The ARP flow will timeout (removed from the switch) if the flow doesn't forward a packet for *x* seconds
        - This field is only relevant when "is-proactive-flood-mode" is set to "false"
    - arp-flow-hard-timeout
        - The ARP flow will timeout (removed from the switch) after *arp-flow-hard-timeout* seconds, regardless of how many packets it is forwarding
        - This field is only relevant when "is-proactive-flood-mode" is set to "false"
- Address Tracker (56-addresstracker.xml)
    - timestamp-update-interval
        - A last-seen timestamp is associated with each address. This last-seen timestamp will only be updated after *timestamp-update-interval* milliseconds.
        - A higher value has the advantage of performing less writes to the database.
        - A lower value has the advantage of knowing how fresh an address is.
    - observe-addresses-from
        - IP and MAC addresses can be observed/learned from ARP, IPv4, and IPv6 packets. Set which packets to make these observations from.
- L2Switch Main (58-l2switchmain.xml)
    - is-install-dropall-flow
        - "true" means a drop-all flow will be installed on each switch, so the default action will be to drop a packet instead of sending it to the controller

    - "false" means this flow will not be installed
- dropall-flow-table-id
    - The dropall flow will be installed on the specified flow table of each switch
    - This field is only relevant when "is-install-dropall-flow" is set to "true"
- dropall-flow-priority
    - The dropall flow will be installed with the specified priority
    - This field is only relevant when "is-install-dropall-flow" is set to "true"
- dropall-flow-idle-timeout
    - The dropall flow will timeout (removed from the switch) if the flow doesn't forward a packet for *x* seconds
    - This field is only relevant when "is-install-dropall-flow" is set to "true"
- dropall-flow-hard-timeout
    - The dropall flow will timeout (removed from the switch) after *x* seconds, regardless of how many packets it is forwarding
    - This field is only relevant when "is-install-dropall-flow" is set to "true"
- is-learning-only-mode
    - "true" means that the L2Switch will only be learning addresses. No additional flows to optimize network traffic will be installed.
    - "false" means that the L2Switch will react to network traffic and install flows on the switches to optimize traffic. Currently, MAC-to-MAC flows are installed.
- reactive-flow-table-id
    - The reactive flow will be installed on the specified flow table of each switch
    - This field is only relevant when "is-learning-only-mode" is set to "false"
- reactive-flow-priority
    - The reactive flow will be installed with the specified priority
    - This field is only relevant when "is-learning-only-mode" is set to "false"
- reactive-flow-idle-timeout
    - The reactive flow will timeout (removed from the switch) if the flow doesn't forward a packet for *x* seconds
    - This field is only relevant when "is-learning-only-mode" is set to "false"

- reactive-flow-hard-timeout
  - The reactive flow will timeout (removed from the switch) after *x* seconds, regardless of how many packets it is forwarding
  - This field is only relevant when "is-learning-only-mode" is set to "false"

9. ODL-SDNi

The User Guide for ODL-SDNi can be found on the OpenDaylight wiki here: https://wiki.opendaylight.org/view/ODL-SDNiApp:User_Guide

10. Packet Cable MultiMedia (PCMM) Service

Table of Contents

Packet Cable MultiMedia (PCMM) provides an interface to control and management service flow for CMTS network elements. A service flows constitute a DOCSIS data path between a CMTS and a subscriber's cable modem (CM) guaranteed application specific quality of service (QoS), known as Dynamic Quality of Service (DQoS). PCMM offers (MSOs) the ability to deliver new services using existing cable infrastructure. MSOs have already begun to apply PCMM technology for expanding their multimedia service offerings.

Overview

The PCMM architecture comprises the following components:

- The Application Manager, which specifies QoS requirements to the Policy Server on a per-application basis.
- The Policy Server, which allocates network resources per subscriber and per application, ensuring that consumption meets MSO priorities.
- The Cable Modem Termination System (CMTS), which enforces policies according to bandwidth capacity.
- The Cable Modem, which resides on the client side and connects the client's network to the cable system.

PacketCable Multimedia defines a service delivery framework that provides general-purpose QoS, event-based accounting, and security functionality founded upon the mechanisms defined in PacketCable 1.x. However, due to the broader spectrum of applications and services addressed by this initiative, each of these functional areas has been revisited and generalized for the present purposes. Telephony-specific requirements and interfaces (e.g., call signaling, PSTN interconnection and electronic surveillance) are not part of PacketCable Multimedia, while core functionality such as QoS resource management mechanisms, has been enhanced. Throughout this process, one of the primary objectives of this work has been to leverage and reuse as much of the existing body of PacketCable 1.x investment, knowledge base, and technical functionality as possible. Key features of the described Multimedia service delivery framework include:

- Simple, powerful access to DOCSIS QoS mechanisms supporting both time and volume-based network resource authorizations,
- Abstract, event-based network resource auditing and management mechanisms,

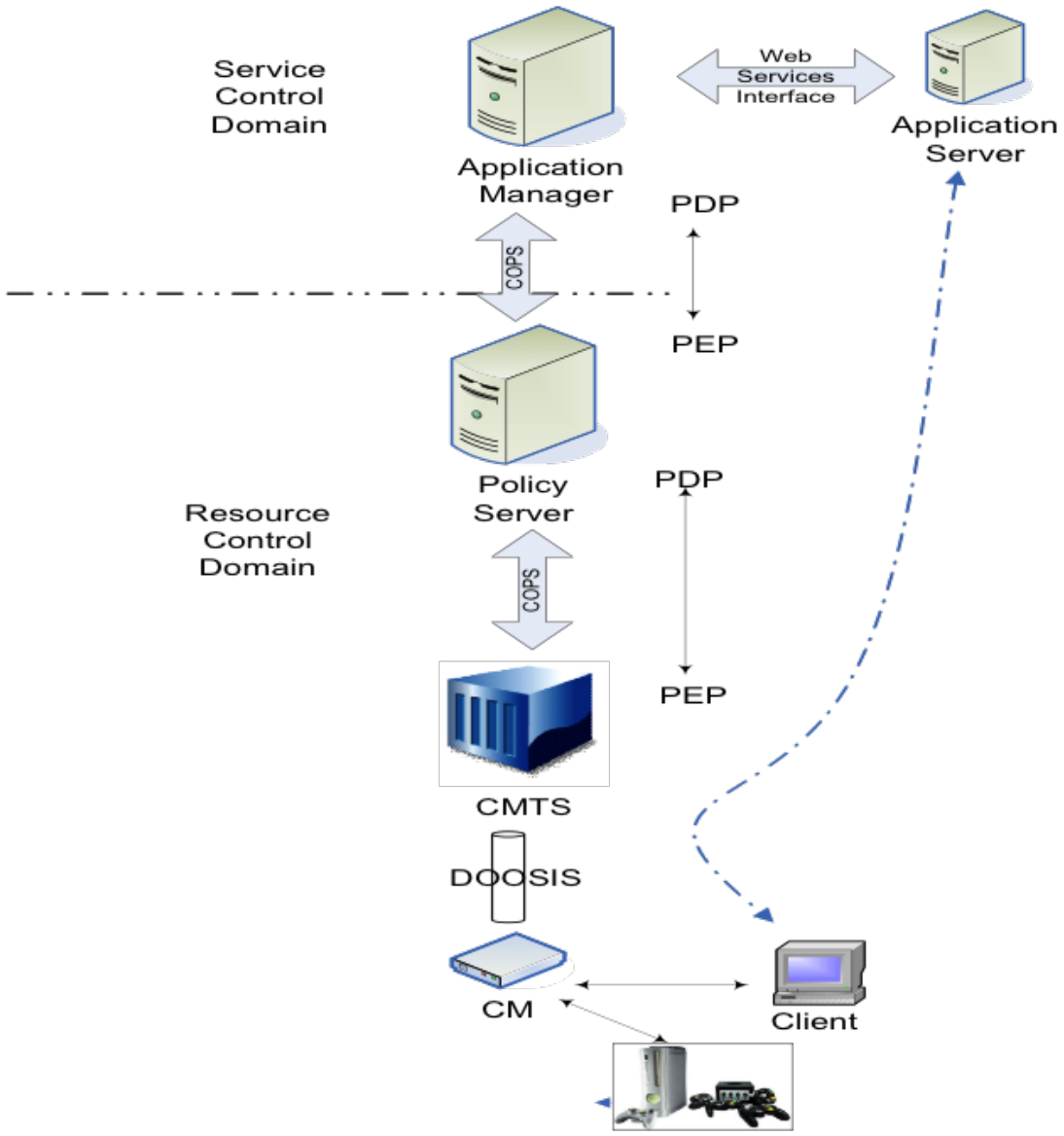
- A robust security infrastructure that provides integrity and appropriate levels of protection across all interfaces.

The goal of this project is to utilizes the OpenDayLight controller platform as for the Application Manager and parts of the Policy Server and leverage the as many existing components offered by the platform.

The initial southbound transport has been written to the following version of the specification: http://www.cablelabs.com/wp-content/uploads/specdocs/PKT-SP-MM-I05-091029.pdf

Architecture

Figure 10.1. Architecture Overview



The OpenDaylight Packetcable PCMM includes:

- Packetcable PCMM Provider
- Packetcable PCMM Consumer
- Packetcable PCMM Model
- Southbound ODL plugin supporting PCMM/COPS protocol driver
- Packetcable PCMM RESTCONF Service API

Features

A brief description of some of what this feature has to offer:

- Provision a CMTS
- Flow Programmer match-only for managing DOCSIS (service) flows
- RESTCONF APIs for provisioning CMTS network elements
- HTML Provisioning Interface and some Python examples
- RESTCONF APIs for provisioning Service Flow values and types
- RESTCONF APIs for provisioning QoS (or metering) parameters
- SAL extensions for DOCSIS specific data model and configuration APIs
- PCMM/COPS protocol transport plugin

Install

```
opendaylight-user@root>feature:install odl-packetcable-all
```

Support

For support please contact the packetcable project at:

- PCMM PacketCable mailing list: dev@lists.opendaylight.org

11. Plugin for OpenContrail

The User Guide for the Plugin for OpenContrail can be found on the OpenDaylight wiki here: https://wiki.opendaylight.org/view/Southbound_Plugin_to_the_OpenContrail_Platform:User_Guide

12. TCP-MD5

The User Guide for TCP-MD5 can be found on the OpenDaylight wiki here: https://wiki.opendaylight.org/view/TCPMD5:Helium_User_Guide