



OPEN
DAYLIGHT

OpenDaylight User Guide



OPEN
DAYLIGHT

OpenDaylight User Guide

Table of Contents

I. Getting Started with Opendaylight	1
1. OpenDaylight Controller Overview	2
2. Using the OpenDaylight User Interface (DLUX)	3
Getting Started with DLUX	3
Logging In	4
Working with DLUX	4
Viewing Network Statistics	5
Viewing Network Topology	5
Interacting with the Open Daylight Controller (ODL)	6
3. Running XSQL Console Commands and Queries	12
XSQL Overview	12
Installing XSQL	12
XSQL Console Commands	12
XSQL Queries	13
4. Setting Up Clustering on an OpenDaylight Controller	15
Clustering Overview	15
Single Node Clustering	15
Multiple Node Clustering	16
II. Addons	22
5. BGP LS PCEP	23
BGP LS	23
PCEP	26
6. Defense4All	30
Defense4All Overview	30
Defense4All User Interface	31
7. Group-Based Policy	36
Architecture and Model	36
Tutorial	36
Contact Information	46
8. L2Switch	47
Running the L2Switch project	47
Create a network using mininet	47
Generating network traffic using mininet	48
Checking Address Observations	48
Checking Hosts	48
Checking STP status of each link	49
Miscellaneous mininet commands	50
Components of the L2Switch	50
Configuration of L2Switch Components	51
9. ODL-SDNi	56
10. Packet Cable MultiMedia (PCMM) Service	57
Overview	57
Architecture	58
Features	59
Support	59
11. Plugin for OpenContrail	60
12. TCP-MD5	61
13. VTN	62

VTN Overview	62
VTN Installation Guide	70
How to set up OpenStack for the integration with VTN Manager	72
VTN Usage Examples	93

List of Figures

2.1. DLUX Modules	4
2.2. Topology Module	6
2.3. Yang UI	7
2.4. Yang API Specification	8
2.5. Yang UI API Specification	9
2.6. DLUX Yang Topology	10
2.7. DLUX List Warnings	11
2.8. DLUX List Button1	11
6.1. Defense4All Overview	30
8.1. Address Observations	48
8.2. Hosts	49
8.3. STP status	50
10.1. Architecture Overview	58
13.1. VTN Overview	63
13.2. VTN Construction	64
13.3. VTN Mapping	65
13.4. VTN FlowFilter	68
13.5. VTN API	69
13.6. LAB Setup	73
13.7. Horizon GUI	78
13.8. Hypervisors	79
13.9. Create Network	80
13.10. Step 1	81
13.11. Step 2	82
13.12. Step 3	83
13.13. Instance Creation	84
13.14. Launch Instance	85
13.15. Launch Network	86
13.16. Load All Instances	87
13.17. Instance Console	88
13.18. Ping	89
13.19. EXAMPLE DEMONSTRATING SINGLE CONTROLLER	93
13.20. EXAMPLE DEMONSTRATING MULTIPLE CONTROLLERS	95
13.21. Example that demonstrates vlanmap testing in Mininet Environment	97
13.22. EXAMPLE DEMONSTRATING VTN STATIONS	99
13.23. Flow Filter	103
13.24. PathMap	106

List of Tables

3.1. Supported XSQL Console Commands	12
3.2. Supported XSQL Query Criteria Operators	13

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

Getting Started with DLUX	3
Logging In	4
Working with DLUX	4
Viewing Network Statistics	5
Viewing Network Topology	5
Interacting with the Open Daylight Controller (ODL)	6

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:

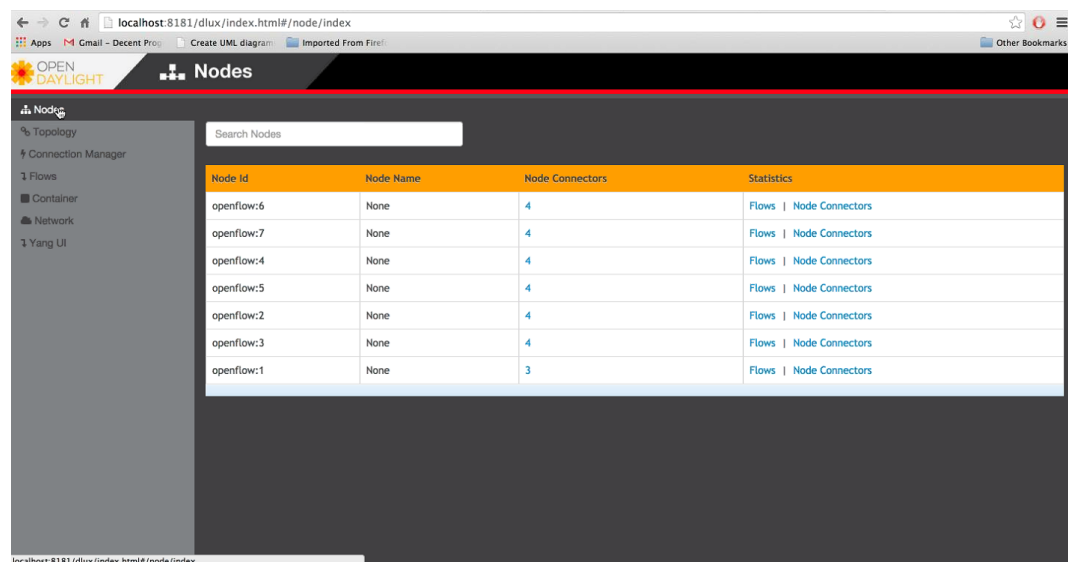
1. Open a browser and enter the login URL. If you have installed DLUX as a stand-alone, then the login URL is <http://localhost:9000/DLUX/index.html>. However if you have deployed DLUX with Karaf, then the login URL is <http://<your IP>:8181/dlux/index.html>. NOTE: Ensure that you use the port applicable 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 I2 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



Node Id	Node Name	Node Connectors	Statistics
openflow:6	None	4	Flows Node Connectors
openflow:7	None	4	Flows Node Connectors
openflow:4	None	4	Flows Node Connectors
openflow:5	None	4	Flows Node Connectors
openflow:2	None	4	Flows Node Connectors
openflow:3	None	4	Flows Node Connectors
openflow:1	None	3	Flows Node Connectors

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 a table 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.



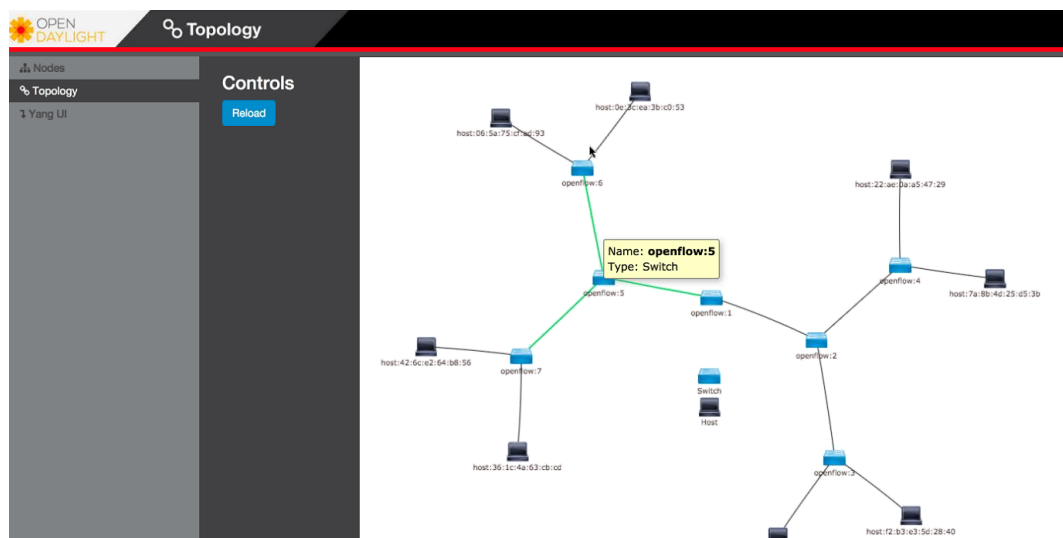
Note

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.

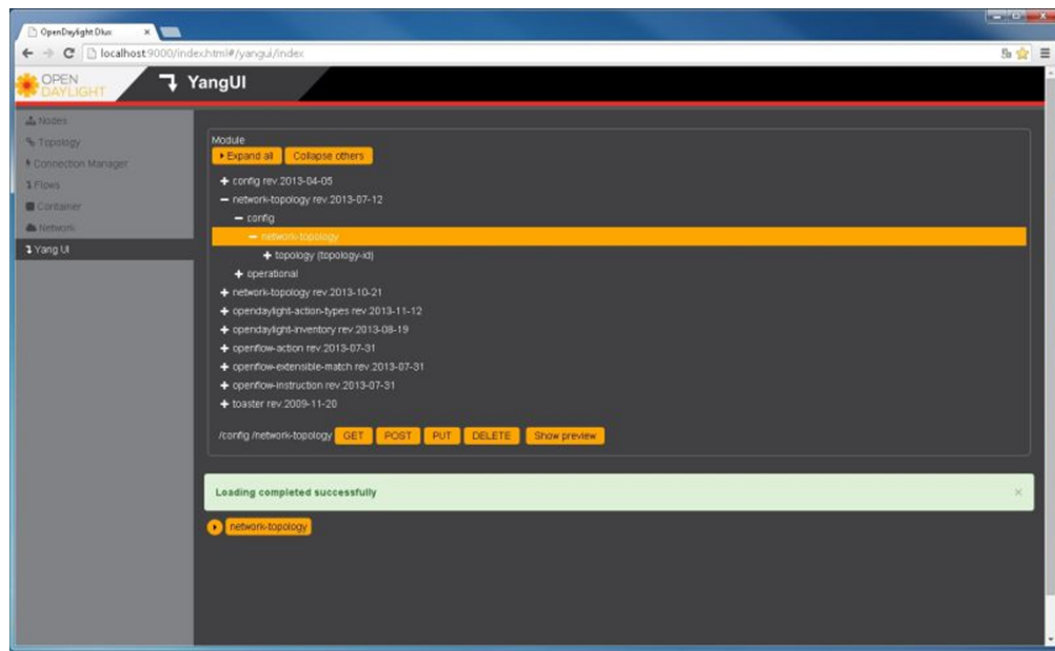
Figure 2.2. Topology Module



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].

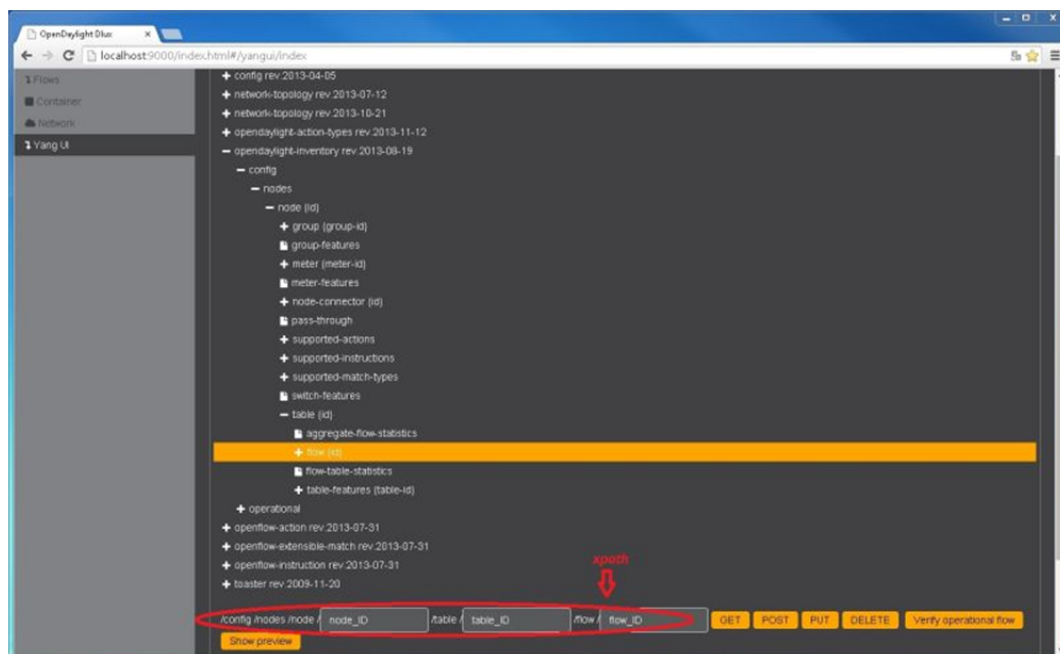
Figure 2.3. Yang UI



To use Yang UI:

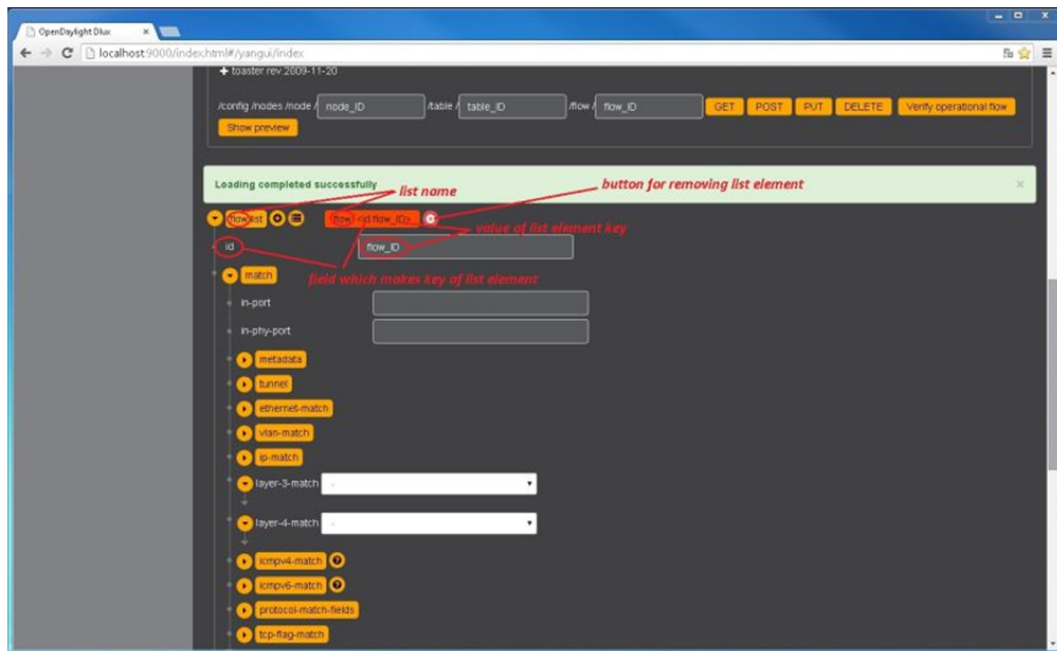
1. Select **Yang UI** on the left pane. The right pane is divided in two parts.
2. The top part displays a tree of APIs and subAPIs and buttons to call possible functions (GET, POST, PUT, DELETE, ...). Not every subAPIs can call every function. For example, subAPIs "operational" have GET functionality 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



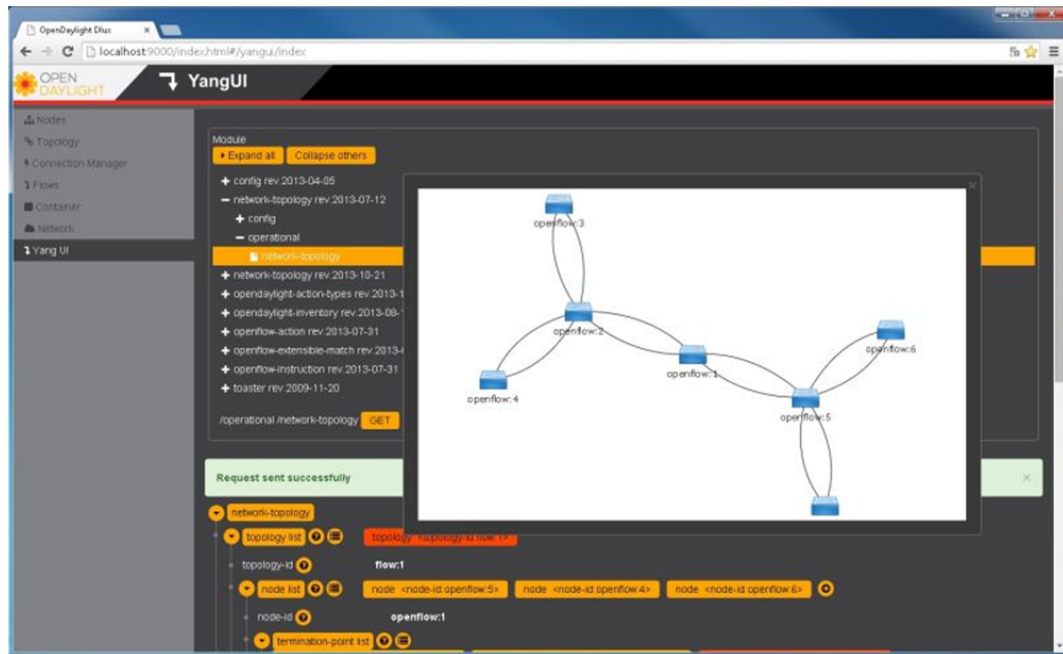
4. Click **Show Preview** button under API tree to display request that will be sent to ODL. A pane is displayed on the right side with text of request when some input is filled.

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

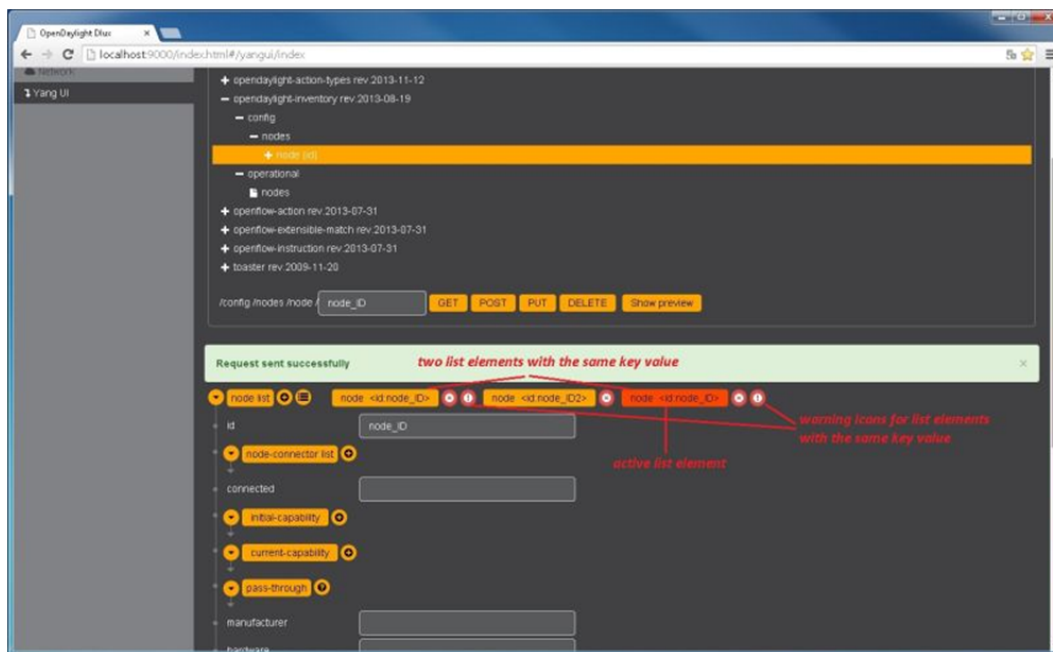
The list is displayed like tree structure with possibility to expand or collapse by the arrow before name of the list. To configure list elements on the Yang UI:

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]

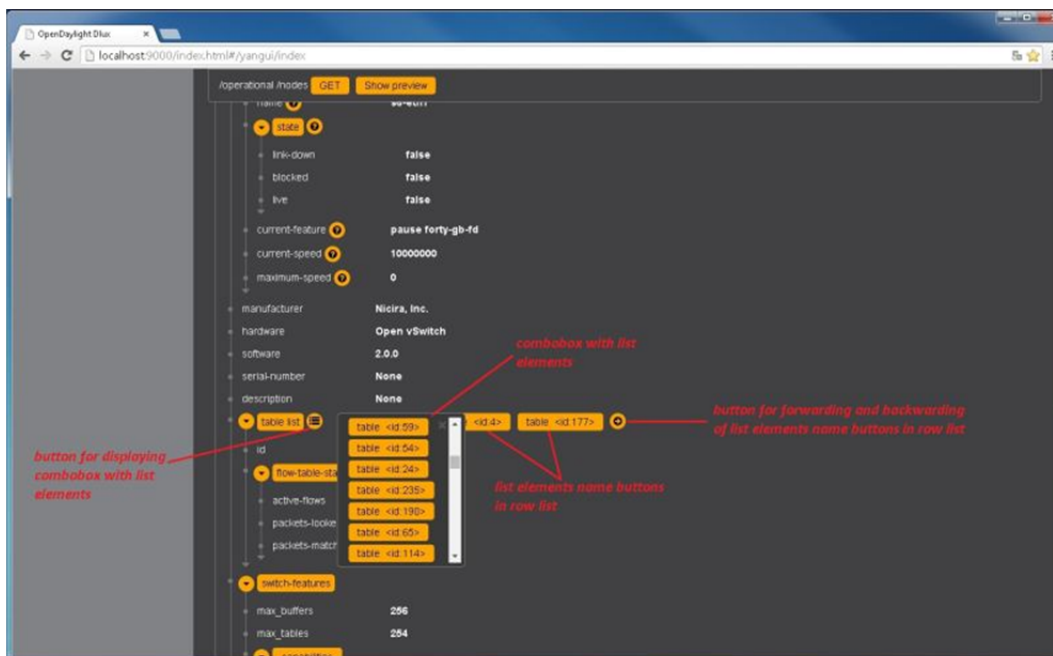
3. Key of list is one or more inputs, which are used like identifier of list element. All list elements in one list must have different key values. If some elements has the same key values, the new warning icon ! is displayed near their name buttons.

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 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	12
Installing XSQL	12
XSQL Console Commands	12
XSQL Queries	13

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-xsq`

XSQL Console Commands

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

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

Table 3.1. Supported XSQL Console Commands

Command	Description
<code>r</code>	Repeats the last command you executed.
<code>list vttables</code>	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 tocsv 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	15
Single Node Clustering	15
Multiple Node Clustering	16

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 its 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 placed 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.

- 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"]
```

- 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"
]
```

- 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>

- 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	23
PCEP	26

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

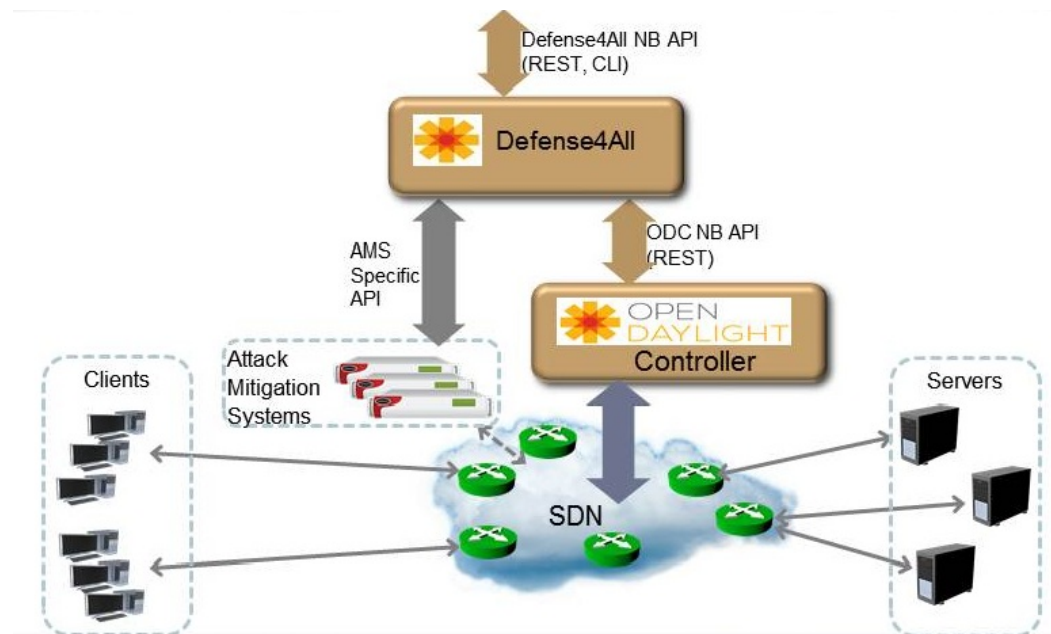
Table of Contents

Defense4All Overview	30
Defense4All User Interface	31

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: sdenablednative.
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** field, 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 Query**. The results display at the bottom of the pane.
- To save the query to a file, enter a file path in the **Filename** field, 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

Architecture and Model	36
Tutorial	36
Contact Information	46

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": "leaf9a67-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-leaf9a67-a171-42a8-9282-71cf702f61dd-22282cca-9a13-4d0c-a67e-
a933ebb0b0ae"
      }
    ],
    "id": "leaf9a67-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-leaf9a67-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": "leaf9a67-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": "leaf9a67-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

Running the L2Switch project	47
Create a network using mininet	47
Generating network traffic using mininet	48
Checking Address Observations	48
Checking Hosts	48
Checking STP status of each link	49
Miscellaneous mininet commands	50
Components of the L2Switch	50
Configuration of L2Switch Components	51

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.

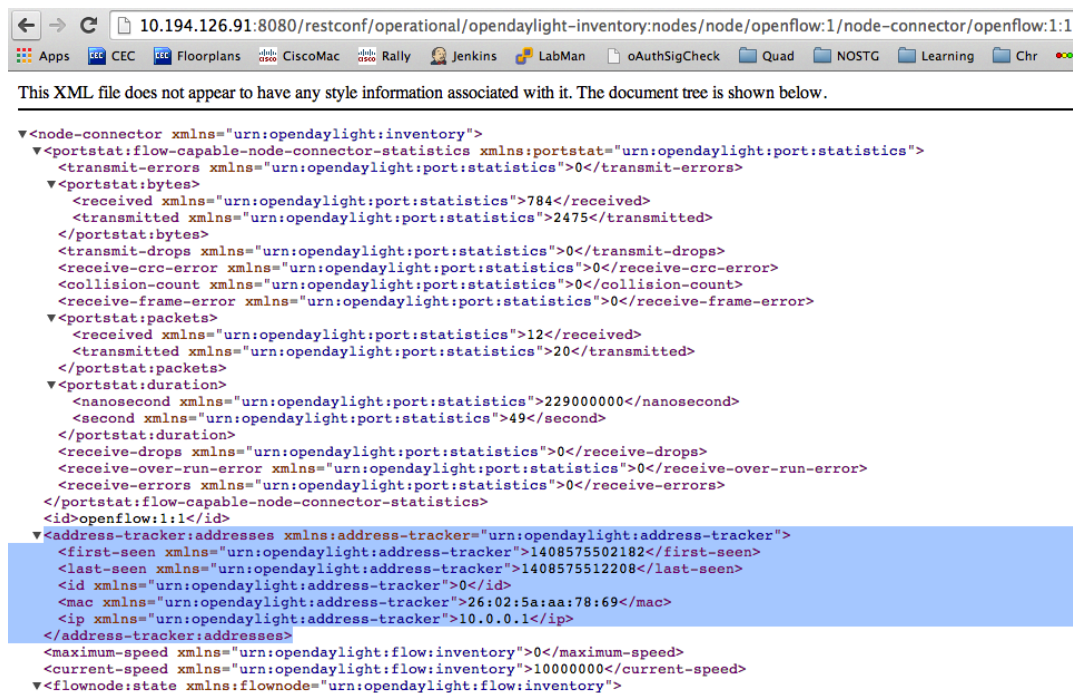
Checking Address Observations

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

```

This XML file does not appear to have any style information associated with it. The document tree is shown below.
<?xml version="1.0" encoding="UTF-8" standalone="no" >
<topology xmlns="urn:TBD:params:xml:ns:yang:network-topology">
  <topology-id>flow:1</topology-id>
  <node>
    <host-track:attachment-points xmlns:host-track="urn:opendaylight:host-tracker">
      <tp-id xmlns="urn:opendaylight:host-tracker">openflow:2:1</tp-id>
    </host-track:attachment-points>
    <host-track:addresses xmlns:host-track="urn:opendaylight:host-tracker">
      <mac xmlns="urn:opendaylight:host-tracker">e6:42:3f:4b:80:5d</mac>
      <first-seen xmlns="urn:opendaylight:host-tracker">1408583775527</first-seen>
      <ip xmlns="urn:opendaylight:host-tracker">10.0.0.2</ip>
      <id xmlns="urn:opendaylight:host-tracker">1</id>
      <last-seen xmlns="urn:opendaylight:host-tracker">1408583780536</last-seen>
    </host-track:addresses>
    <id xmlns="urn:opendaylight:host-tracker">e6:42:3f:4b:80:5d</id>
  </node>
  <node>...</node>
  <node>...</node>
  <node>
    <inventory-node-ref xmlns:wxtj="urn:opendaylight:inventory" xmlns="urn:opendaylight:model:top">
      <node-id>openflow:1</node-id>
    </node>
    <link>
      <destination>
        <dest-tp>openflow:1:2</dest-tp>
        <dest-node>openflow:1</dest-node>
      </destination>
      <source>
        <source-node>openflow:2</source-node>
        <source-tp>openflow:2:2</source-tp>
      </source>
      <link-id>openflow:2:2</link-id>
    </link>
  </node>
</topology>

```

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

```

This XML file does not appear to have any style information associated with it. The document tree is shown below.

<node-connector xmlns="urn:.opendaylight:inventory">
  <portstat:flow-capable-node-connector-statistics xmlns:portstat="urn:.opendaylight:port:statistics">
    <transmit-errors xmlns="urn:.opendaylight:port:statistics">0</transmit-errors>
    <portstat:bytes>
      <received xmlns="urn:.opendaylight:port:statistics">28054</received>
      <transmitted xmlns="urn:.opendaylight:port:statistics">2856</transmitted>
    </portstat:bytes>
    <transmit-drops xmlns="urn:.opendaylight:port:statistics">0</transmit-drops>
    <receive-crc-error xmlns="urn:.opendaylight:port:statistics">0</receive-crc-error>
    <collision-count xmlns="urn:.opendaylight:port:statistics">0</collision-count>
    <receive-frame-error xmlns="urn:.opendaylight:port:statistics">0</receive-frame-error>
    <portstat:packets>
      <received xmlns="urn:.opendaylight:port:statistics">153</received>
      <transmitted xmlns="urn:.opendaylight:port:statistics">46</transmitted>
    </portstat:packets>
    <portstat:duration>
      <nanosecond xmlns="urn:.opendaylight:port:statistics">41500000</nanosecond>
      <second xmlns="urn:.opendaylight:port:statistics">218</second>
    </portstat:duration>
    <receive-drops xmlns="urn:.opendaylight:port:statistics">0</receive-drops>
    <receive-over-run-error xmlns="urn:.opendaylight:port:statistics">0</receive-over-run-error>
    <receive-errors xmlns="urn:.opendaylight:port:statistics">0</receive-errors>
  </portstat:flow-capable-node-connector-statistics>
  <id>openflow:1:2</id>
  <status xmlns="urn:.opendaylight:l2switch:loopremover">forwarding</status>
  <maximum-speed xmlns="urn:.opendaylight:flow:inventory">0</maximum-speed>
  <current-speed xmlns="urn:.opendaylight:flow:inventory">10000000</current-speed>

```

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

Overview	57
Architecture	58
Features	59
Support	59

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,

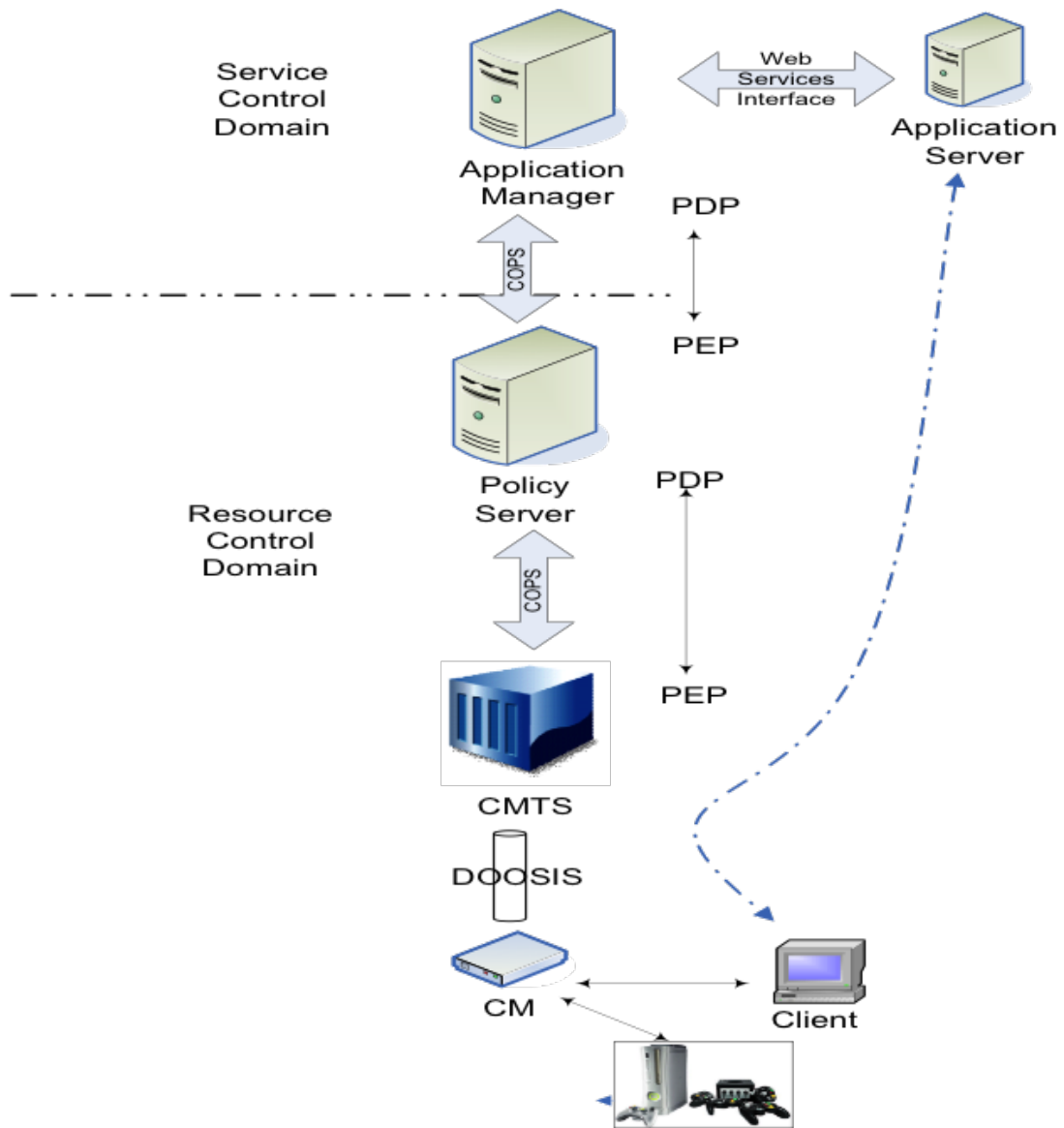
- A robust security infrastructure that provides integrity and appropriate levels of protection across all interfaces.

The goal of this project is to utilize 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-105-091029.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

13. VTN

Table of Contents

VTN Overview	62
VTN Installation Guide	70
How to set up OpenStack for the integration with VTN Manager	72
VTN Usage Examples	93

VTN Overview

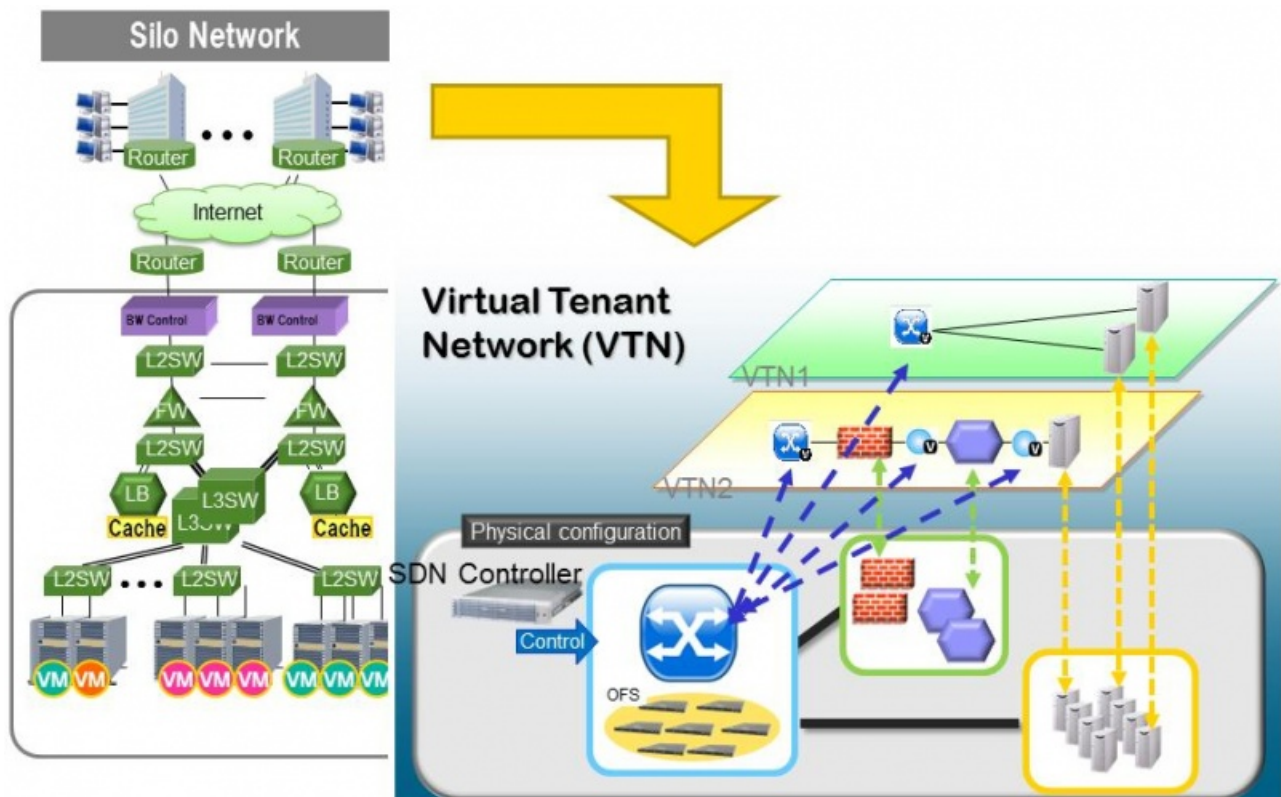
OpenDaylight Virtual Tenant Network (VTN) is an application that provides multi-tenant virtual network on an SDN controller.

Conventionally, huge investment in the network systems and operating expenses are needed because the network is configured as a silo for each department and system. So, various network appliances must be installed for each tenant and those boxes cannot be shared with others. It is a heavy work to design, implement and operate the entire complex network.

The uniqueness of VTN is a logical abstraction plane. This enables the complete separation of logical plane from physical plane. Users can design and deploy any desired network without knowing the physical network topology or bandwidth restrictions.

VTN allows the users to define the network with a look and feel of conventional L2/L3 network. Once the network is designed on VTN, it will automatically be mapped into underlying physical network, and then configured on the individual switch leveraging SDN control protocol. The definition of logical plane makes it possible not only to hide the complexity of the underlying network but also to better manage network resources. It achieves reducing reconfiguration time of network services and minimizing network configuration errors.

Figure 13.1. VTN Overview



Network Virtualization Function

The user first defines a VTN. Then, the user maps the VTN to a physical network, which enables communication to take place according to the VTN definition. With the VTN definition, L2 and L3 transfer functions and flow-based traffic control functions (filtering and redirect) are possible.

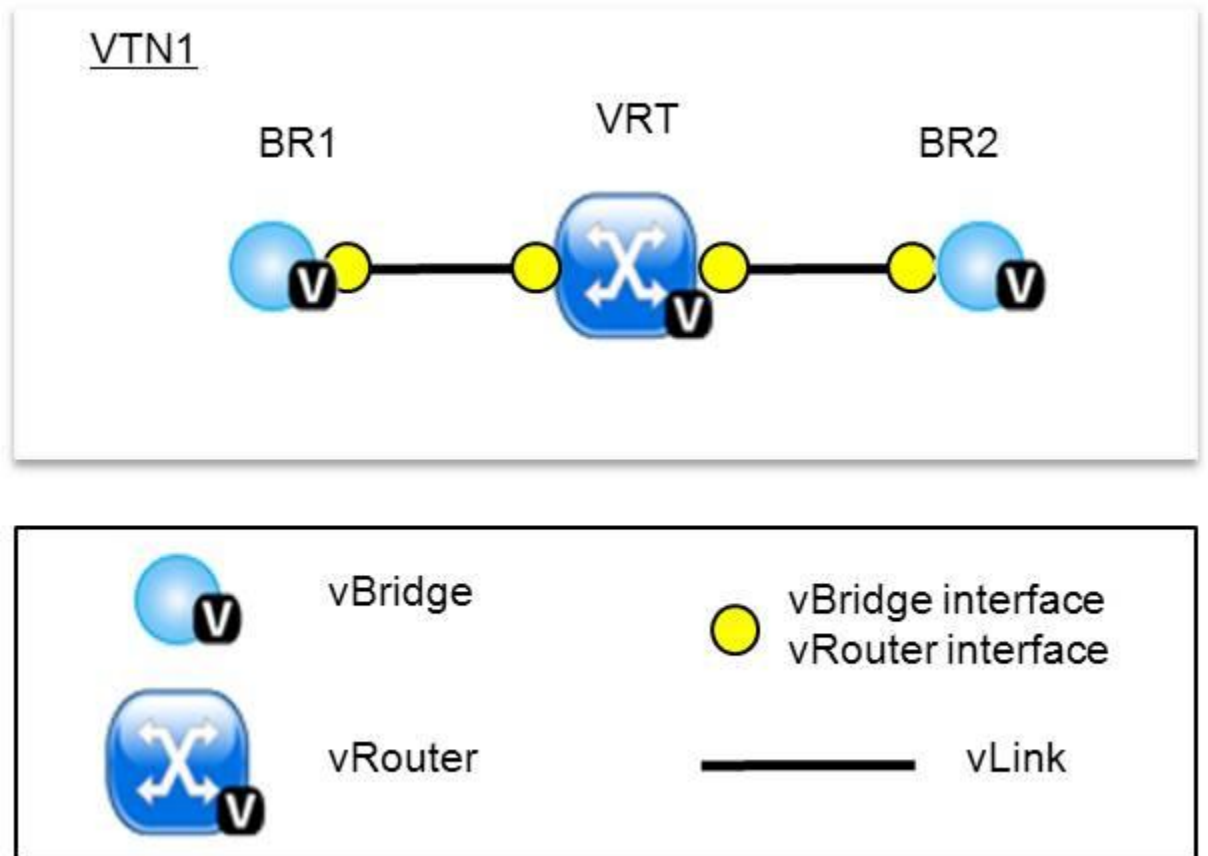
Virtual Network Construction

The following table shows the elements which make up the VTN. In the VTN, a virtual network is constructed using virtual nodes (vBridge, vRouter) and virtual interfaces and links. It is possible to configure a network which has L2 and L3 transfer function, by connecting the virtual interfaces made on virtual nodes via virtual links.

vBridge	The logical representation of L2 switch function.
vRouter	The logical representation of router function.
vTep	The logical representation of Tunnel End Point - TEP.
vTunnel	The logical representation of Tunnel.
vBypass	The logical representation of connectivity between controlled networks.
Virtual interface	The representation of end point on the virtual node.
Virtual Link(vLink)	The logical representation of L1 connectivity between virtual interfaces.

The following figure shows an example of a constructed virtual network. VRT is defined as the vRouter, BR1 and BR2 are defined as vBridges. interfaces of the vRouter and vBridges are connected using vLinks.

Figure 13.2. VTN Construction



Mapping of Physical Network Resources

Map physical network resources to the constructed virtual network. Mapping identifies which virtual network each packet transmitted or received by an OpenFlow switch belongs to, as well as which interface in the OpenFlow switch transmits or receives that packet. There are two mapping methods. When a packet is received from the OFS, port mapping is first searched for the corresponding mapping definition, then VLAN mapping is searched, and the packet is mapped to the relevant vBridge according to the first matching mapping.

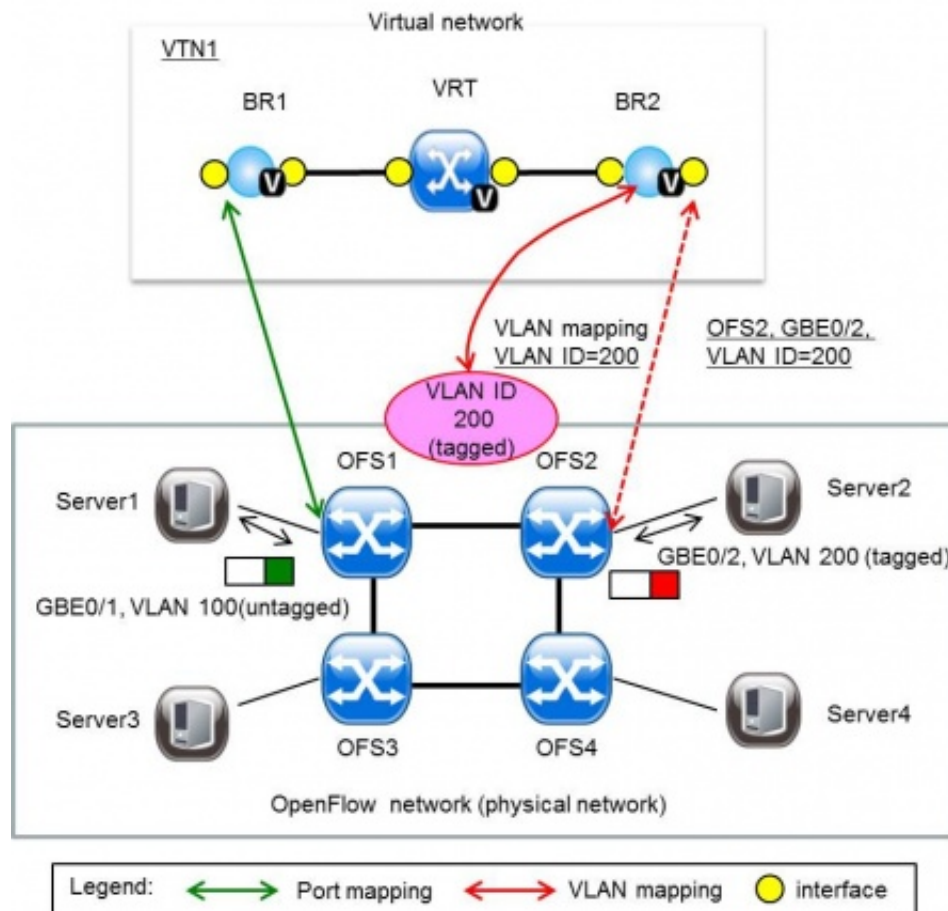
Port mapping	Maps physical network resources to an interface of vBridge using Switch ID, Port ID and VLAN ID of the incoming L2 frame. Untagged frame mapping is also supported.
VLAN mapping	Maps physical network resources to a vBridge using VLAN ID of the incoming L2 frame. Maps physical resources of a particular switch to a vBridge using switch ID and VLAN ID of the incoming L2 frame.
MAC mapping	Maps physical resources to an interface of vBridge using MAC address of the incoming L2 frame (The initial contribution does not include this method).

VTN can learn the terminal information from a terminal that is connected to a switch which is mapped to VTN. Further, it is possible to refer that terminal information on the VTN.

- Learning terminal information VTN learns the information of a terminal that belongs to VTN. It will store the MAC address and VLAN ID of the terminal in relation to the port of the switch.
- Aging of terminal information Terminal information, learned by the VTN, will be maintained until the packets from terminal keep flowing in VTN. If the terminal gets disconnected from the VTN, then the aging timer will start clicking and the terminal information will be maintained till timeout.

The following figure shows an example of mapping. An interface of BR1 is mapped to port GBE0/1 of OFS1 using port mapping. BR2 is mapped to VLAN 200 using VLAN mapping. Packets received from GBE0/1 of OFS1 are regarded as those from the corresponding interface of BR1. BR2 is mapped to VLAN 200 using VLAN mapping. Packets with VLAN tag 200 received from any ports of any OFSs are regarded as those from an interface of BR2.

Figure 13.3. VTN Mapping



vBridge Functions

The vBridge provides the bridge function that transfers a packet to the intended virtual port according to the destination MAC address. The vBridge looks up the MAC address

table and transmits the packet to the corresponding virtual interface when the destination MAC address has been learned. When the destination MAC address has not been learned, it transmits the packet to all virtual interfaces other than the receiving port (flooding). MAC addresses are learned as follows.

- **MAC address learning** The vBridge learns the MAC address of the connected host. The source MAC address of each received frame is mapped to the receiving virtual interface, and this MAC address is stored in the MAC address table created on a per-vBridge basis.
- **MAC address aging** The MAC address stored in the MAC address table is retained as long as the host returns the ARP reply. After the host is disconnected, the address is retained until the aging timer times out. To have the vBridge learn MAC addresses statically, you can register MAC addresses manually.

vRouter Functions

The vRouter transfers IPv4 packets between vBridges. The vRouter supports routing, ARP learning, and ARP aging functions. The following outlines the functions.

- **Routing function** When an IP address is registered with a virtual interface of the vRouter, the default routing information for that interface is registered. It is also possible to statically register routing information for a virtual interface.
- **ARP learning function** The vRouter associates a destination IP address, MAC address and a virtual interface, based on an ARP request to its host or a reply packet for an ARP request, and maintains this information in an ARP table prepared for each routing domain. The registered ARP entry is retained until the aging timer, described later, times out. The vRouter transmits an ARP request on an individual aging timer basis and deletes the associated entry from the ARP table if no reply is returned. For static ARP learning, you can register ARP entry information manually. *DHCP relay agent function
The vRouter also provides the DHCP relay agent function.

Flow Filter Functions

Flow Filter function is similar to ACL. It is possible to allow or prohibit communication with only certain kind of packets that meet a particular condition. Also, it can perform a processing called Redirection - WayPoint routing, which is different from the existing ACL. Flow Filter can be applied to any interface of a vNode within VTN, and it is possible to control the packets that pass interface. The match conditions that could be specified in Flow Filter are as follows. It is also possible to specify a combination of multiple conditions.

- Source MAC address
- Destination MAC address
- MAC ether type
- VLAN Priority
- Source IP address
- Destination IP address

- DSCP
- IP Protocol
- TCP/UDP source port
- TCP/UDP destination port
- ICMP type
- ICMP code

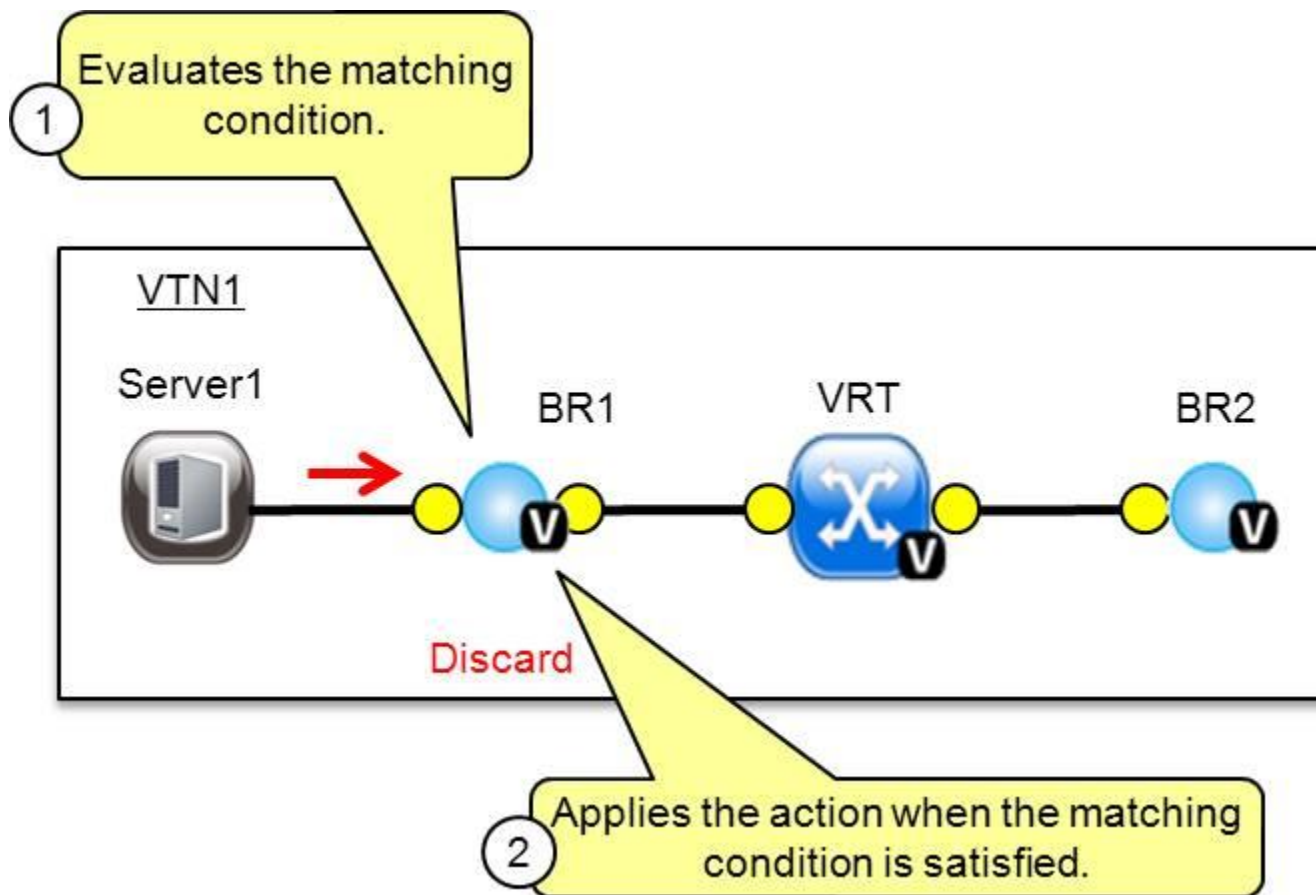
The types of Action that can be applied on packets that match the Flow Filter conditions are given in the following table. It is possible to make only those packets, which match a particular condition, to pass through a particular server by specifying Redirection in Action. E.g., path of flow can be changed for each packet sent from a particular terminal, depending upon the destination IP address. VLAN priority control and DSCP marking are also supported.

Pass	Pass particular packets matching the specified conditions.
Drop	Discards particular packets matching the specified conditions.
Redirection	Redirects the packet to a desired virtual interface. Both Transparent Redirection (not changing MAC address) and Router Redirection (changing MAC address) are supported.

The following figure shows an example of how the flow filter function works.

If there is any matching condition specified by flow filter when a packet being transferred within a virtual network goes through a virtual interface, the function evaluates the matching condition to see whether the packet matches it.

If the packet matches the condition, the function applies the matching action specified by flow filter. In the example shown in the figure, the function evaluates the matching condition at BR1 and discards the packet if it matches the condition.

Figure 13.4. VTN FlowFilter

Multiple SDN Controller Coordination

With the network abstractions, VTN enables to configure virtual network across multiple SDN controllers. This provides highly scalable network system.

VTN can be created on each SDN controller. If users would like to manage those multiple VTNs with one policy, those VTNs can be integrated to a single VTN.

As a use case, this feature is deployed to multi data center environment. Even if those data centers are geographically separated and controlled with different controllers, a single policy virtual network can be realized with VTN.

Also, one can easily add a new SDN Controller to an existing VTN or delete a particular SDN Controller from VTN.

In addition to this, one can define a VTN which covers both OpenFlow network and Overlay network at the same time.

Flow Filter, which is set on the VTN, will be automatically applied on the newly added SDN Controller.

Coordination between OpenFlow Network and L2/L3 Network

It is possible to configure VTN on an environment where there is mix of L2/L3 switches as well. L2/L3 switch will be shown on VTN as vBypass. Flow Filter or policing cannot be configured for a vBypass. However, it is possible to treat it as a virtual node inside VTN.

Virtual Tenant Network (VTN) API

VTN provides Web APIs. They are implemented by REST architecture and provide the access to resources within VTN that are identified by URI. User can perform the operations like GET/PUT/POST/DELETE against the virtual network resources (e.g. vBridge or vRouter) by sending a message to VTN through HTTPS communication in XML or JSON format.

Figure 13.5. VTN API



Function Outline

VTN provides following operations for various network resources.

Resources	GET	POST	PUT	DELETE
VTN	Yes	Yes	Yes	Yes
vBridge	Yes	Yes	Yes	Yes
vRouter	Yes	Yes	Yes	Yes
vTep	Yes	Yes	Yes	Yes
vTunnel	Yes	Yes	Yes	Yes
vBypass	Yes	Yes	Yes	Yes
vLink	Yes	Yes	Yes	Yes
Interface	Yes	Yes	Yes	Yes
Port map	Yes	No	Yes	Yes
Vlan map	Yes	Yes	Yes	Yes
Flowfilter (ACL/redirect)	Yes	Yes	Yes	Yes
Controller information	Yes	Yes	Yes	Yes
Physical topology information	Yes	No	No	No
Alarm information	Yes	No	No	No

(Example) Connecting the terminal to virtual network

The following is an example of the usage to connect the terminal to the network.

- Create VTN

```
curl -X POST -H 'content-type: application/json' -H 'username: admin' -H
'password: PASSWORD' -H 'ipaddr: 127.0.0.1' \
-d '{"vtn":{"vtn_name":"VTN1"}}' http://172.1.0.1:8080/vtn-webapi/vtns.json
```

- Create Controller Information

```
curl -X POST -H 'content-type: application/json' -H 'username: admin' -H
'password: PASSWORD' -H 'ipaddr: 127.0.0.1' \
-d '{"controller":{"controller_id":"CONTROLLER1","ipaddr":"172.1.0.1",
"type":"pfc","username":"root", \
"password":"PASSWORD","version":"5.0"}}' http://172.1.0.1:8080/vtn-webapi/
controllers.json
```

- Create vBridge under VTN

```
curl -X POST -H 'content-type: application/json' -H 'username: admin' -H
'password: PASSOWRD' -H 'ipaddr: 127.0.0.1' \
-d '{"vbridge":{"vbr_name":"VBR1","controller_id": "CONTROLLER1",
"domain_id": "(DEFAULT)"}' \
http://172.1.0.1:8080/vtn-webapi/vtns/VTN1/vbridges.json
```

- Create the interface to connect the terminal under vBridge

```
curl -X POST -H 'content-type: application/json' -H 'username: admin' -H
'password: PASSWORD' -H 'ipaddr: 127.0.0.1' \
-d '{"interface":{"if_name":"IF1"}}' http://172.1.0.1:8080/vtn-webapi/vtns/
VTN1/vbridges/VBR1/interfaces.json
```

VTN Installation Guide

This guide explains how to install VTN Coordinator and VTN Manager to use VTN API.

VTN Manager is a set of OSGi bundles running in OpenDaylight controller, and VTN Coordinator is an application running outside the controller.

It is recommended to install VTN Coordinator and OpenDaylight controller on different machines, since their supported platforms and prerequisites are different.

Supported Platforms

Software	Supported Platforms
VTN Manager	All Java Platform
VTN Coordinator	(64-bit version of below Linux variants) Fedora 20 CentOS 6 RHEL 6 RHEL 7 CentOS 7

Downloading Helium

- Go to the download page of.opendaylight.org. <http://www.opendaylight.org/software/downloads>

- Download the "Pre-Built Zip File" or "Pre-Built Tar File" of the latest Helium.
- Extract files from the downloaded file. e.g.

```
unzip distribution-karaf-0.2.1-Helium-SR1.zip
or
tar zxvf distribution-karaf-0.2.1-Helium-SR1.tar.gz
```

Setting up VTN Coordinator

- Arrange a physical/virtual server with any one of the supported 64-bit OS environment listed above.

Installing the Java

- For RHEL/CentOS 6.1 (x86_64) Download Oracle JDK 7 and install it. <http://www.oracle.com/technetwork/java/javase/downloads/index.html>
- For RHEL/CentOS 6.4 (x86_64)

```
Install OpenJDK 7.
yum install java-1.7.0-openjdk-devel
```

- For Fedora 19/20 Install OpenJDK 7

```
yum install java-1.7.0-openjdk-devel
```

Installing prerequisites

- For RHEL 6

```
yum install perl-Digest-SHA uuid libxslt libcurl unixODBC
wget http://dl.fedoraproject.org/pub/epel/6/i386/epel-release-6-8.noarch.rpm
rpm -Uvh epel-release-6-8.noarch.rpm
yum install json-c
```

- For CentOS6 and Fedora 20

```
yum install perl-Digest-SHA uuid libxslt libcurl unixODBC json-c
```

Installing PostgreSQL Database

- Configure Yum repository to download the latest rpms for PostgreSQL 9.1

```
rpm -ivh http://yum.postgresql.org/9.1/redhat/rhel-6-x86_64/pgdg-redhat91-9.1-5.noarch.rpm
```

- Install the required PostgreSQL packages

```
yum install postgresql91-libs postgresql91 postgresql91-server postgresql91-contrib postgresql91-odbc
```



Note

* If you are facing any problems while installing postgresQL rpm, Please refer to: <https://wiki.opendaylight.org/view/>

[OpenDaylight_Virtual_Tenant_Network_\(VTN\):Installation:Troubleshooting#Problems_while_In](#)

* VTN Coordinator support PostgreSQL version greater than 9.1 only and currently tested with 9.1 and 9.3. Please ensure the PostgreSQL version ≥ 9.1 is installed.

Installing the VTN Coordinator

- Enter into the *externalapps* directory in the top directory of Helium.

```
cd distribution-karaf-0.2.1-Helium-SR1/externalapps
```

- Untar the package *distribution.vtn-coordinator-6.0.0.1-Helium-SR1-bin.tar.bz2* to extract VTN Coordinator from the tar.bz2 file in the "externalapps" directory.

This will install VTN Coordinator to "/usr/local/vtn" directory. The name of the tar.bz2 file name varies depending on the version. Please give the same tar.bz2 file name which is there in your directory.



Note

VTN Coordinator runs on port 8083 (TCP) for REST API by default. If you want to run it on different port other than the default, change the port number in the below file:

```
/usr/local/vtn/tomcat/conf/tomcat-env.sh
```

Configuring database for VTN Coordinator

- Execute the below command.

```
/usr/local/vtn/sbin/db_setup
```

How to set up OpenStack for the integration with VTN Manager

This guide describes how to set up OpenStack for integration with OpenDaylight Controller.

While OpenDaylight Controller provides several ways to integrate with OpenStack, this guide focus on the way which uses VTN features available on OpenDaylight controller. In the integration, VTN Manager work as network service provider for OpenStack.

VTN Manager features, enable OpenStack to work in pure OpenFlow environment in which all switches in data plane are OpenFlow switch.

Requirements

To use OpenDaylight Controller (ODL) as Network Service Provider for Openstack.

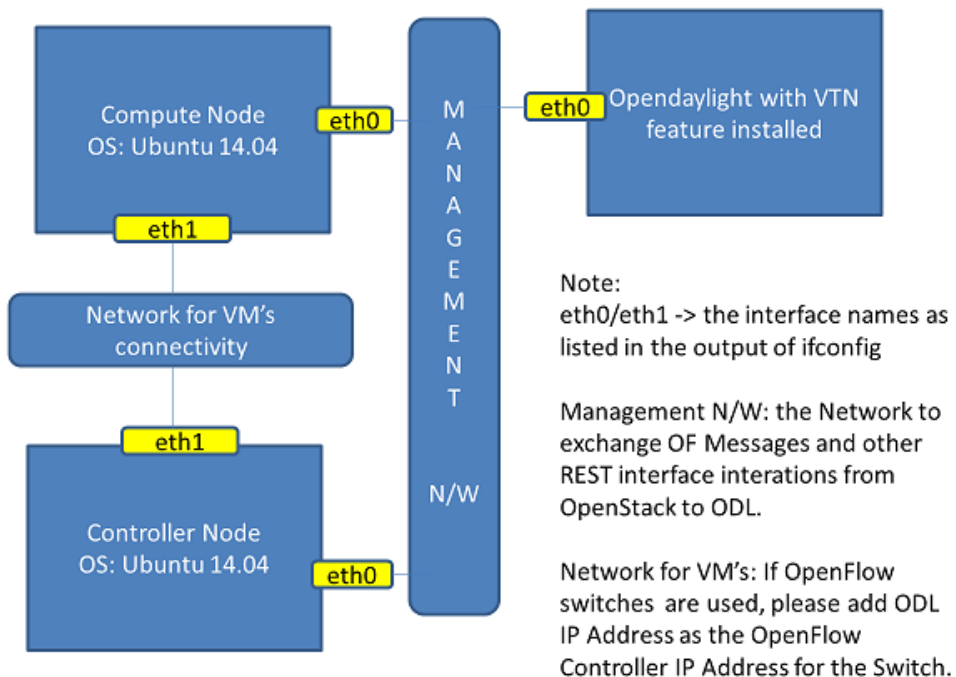
Components

- OpenDaylight Controller.

- OpenStack Control Node.
- OpenStack Compute Node.
- OpenFlow Switch like mininet(Not Mandatory).

The VTN features support multiple OpenStack nodes. You can deploy multiple OpenStack Compute Nodes. In management plane, OpenDaylight Controller, OpenStack nodes and OpenFlow switches should communicate with each other. In data plane, Open vSwitches running in OpenStack nodes should communicate with each other through a physical or logical OpenFlow switches. The core OpenFlow switches are not mandatory. Therefore, you can directly connect to the Open vSwitch's.

Figure 13.6. LAB Setup



Note

Ubuntu 14.04 was used in both the nodes and Vsphere was used for this howto.

Configuration

Server Preparation

- Install Ubuntu 14.04 LTS in two servers (OpenStack Control node and Compute node respectively)

- While installing, Ubuntu mandates creation of a User, we created the user "stack"(We will use the same user for running devstack) NOTE: You can also have multiple Compute nodes. TIP: Please do minimal Install to avoid any issues in devstack bringup

User Settings - Login to both servers - Disable Ubuntu Firewall

```
sudo ufw disable
```

- Optionally install these packages

```
sudo apt-get install net-tools
```

- Edit sudo vim /etc/sudoers and add an entry as follows

```
stack ALL=(ALL) NOPASSWD: ALL
```

Network Settings - Checked the output of `ifconfig -a`, two interfaces were listed `eth0` and `eth1` as indicated in the image above. - We had connected `eth0` interface to the Network where ODL Controller is reachable. - `eth1` interface in both servers were connected to a different network to act as data plane for the VM's created using the OpenStack. - Manually edited the file : `sudo vim /etc/network/interfaces` and made entries as follows

```
stack@ubuntu-devstack:~/devstack$ cat /etc/network/interfaces
# This file describes the network interfaces available on your system
# and how to activate them. For more information, see interfaces(5).
# The loop-back network interface
auto lo
iface lo inet loopback
# The primary network interface
auto eth0
iface eth0 inet static
    address <IP_ADDRESS_TO_REACH_ODL>
    netmask <NET_MASK>
    broadcast <BROADCAST_IP_ADDRESS>
    gateway <GATEWAY_IP_ADDRESS>
auto eth1
iface eth1 inet static
    address <IP_ADDRESS_UNIQ>
    netmask <NETMASK>
```



Note

Please ensure that the `eth0` interface is the default route and it is able to reach the `ODL_IP_ADDRESS` NOTE: The entries for `eth1` are not mandatory, If not set, we may have to manually do "`ifup eth1`" after the stacking is complete to activate the interface

Finalize - reboot both nodes after the user and network settings to have the network settings applied to the network - Login again and check the output of `ifconfig` to ensure that both interfaces are listed

ODL Settings and Execution

vtn.ini

- VTN uses the configuration parameters from `vtn.ini` file for the OpenStack integration.

- These values will be set for the OpenvSwitch, in all the participating OpenStack nodes.
- A configuration file *vtn.ini* needs to be created manually in the *'configuration* directory.
- The contents of *vtn.ini* should be as follows:

```
bridgename=br-int  
portname=eth1  
protocols=OpenFlow13  
failmode=secure
```

- The values of the configuration parameters must be changed based on the user environment.
- Especially, "portname" should be carefully configured, because if the value is wrong, OpenDaylight controller fails to forward packets.
- Other parameters works fine as is for general use cases. bridgename
- The name of the bridge in Open vSwitch, that will be created by OpenDaylight Controller.
- It must be "br-int". portname
- The name of the port that will be created in the vbridge in Open vSwitch.
- This must be the same name of the interface of OpenStack Nodes which is used for interconnecting OpenStack Nodes in data plane.(in our case:eth1)
- By default, if vtn.ini is not created, VTN uses ens33 as portname. protocols
- OpenFlow protocol through which OpenFlow Switch and Controller communicate.
- The values can be OpenFlow13 or OpenFlow10. failmode
- The value can be "standalone" or "secure".
- Please use "secure" for general use cases.

Start ODL Controller

- Please refer to the https://wiki.opendaylight.org/view/Release/Helium/VTN/Installation_Guide to run ODL with VTN Feature enabled.



Tip

After running ODL Controller, please ensure ODL Controller listens to the ports:6633,6653, 6640 and 8080



Tip

Please allow the ports in firewall for the devstack to be able to communicate with ODL Controller.

**Note**

6633/6653 - OpenFlow Ports

**Note**

6640 - OVS Manager Port

**Note**

8080 - Port for REST API

Devstack Setup

Get Devstack (All nodes)

Install git application using

```
sudo apt-get install git
get devstack
git clone https://git.openstack.org/openstack-dev/devstack;
```

Switch to stable/Juno Version branch

```
cd devstack
git checkout stable/juno
```

Stack Control Node

local.conf:

```
cd devstack in the controller node
* Copy the contents of local.conf (devstack control node) and save it as
"local.conf" in the devstack.
* Please modify the IP Address values as required.
* Stack the node
```

```
./stack.sh
```

Verify Control Node stacking

- stack.sh prints out Horizon is now available at http://<CONTROL_NODE_IP_ADDRESS>:8080/
- Execute the command `sudo ovs-vsctl show` in the control node terminal and verify if the bridge `br-int` is created.

Stack Compute Node

- local.conf:
- cd devstack in the controller node

- Copy the contents of local.conf (devstack compute node) and save it as local.conf in the *devstack*".
- Please modify the IP Address values as required.
- Stack the node

```
./stack.sh
```

Verify Compute Node stacking

- stack.sh prints out This is your host ip: <COMPUTE_NODE_IP_ADDRESS>
- Execute the command *sudo ovs-vsctl show* in the control node terminal and verify if the bridge *br-int* is created.
- The output of the ovs-vsctl show will be similar to the one seen in control node. =====
Additional Verifications
- Please visit the ODL DLUX GUI after stacking all the nodes, http://<ODL_IP_ADDRESS>:8181/dlux/index.html. The switches, topology and the ports that are currently read can be validated.



Tip

If the interconnected between the OVS is not seen, Please bring up the interface for the dataplane manually using the below comamnd

```
ifup <interface_name>
```



Tip

Some versions of OVS, drop packets when there is a table-miss, So please add the below flow to all the nodes with OVS version (>=2.1)

```
ovs-ofctl --protocols=OpenFlow13 add-flow br-int priority=0,actions=output:CONTROLLER
```



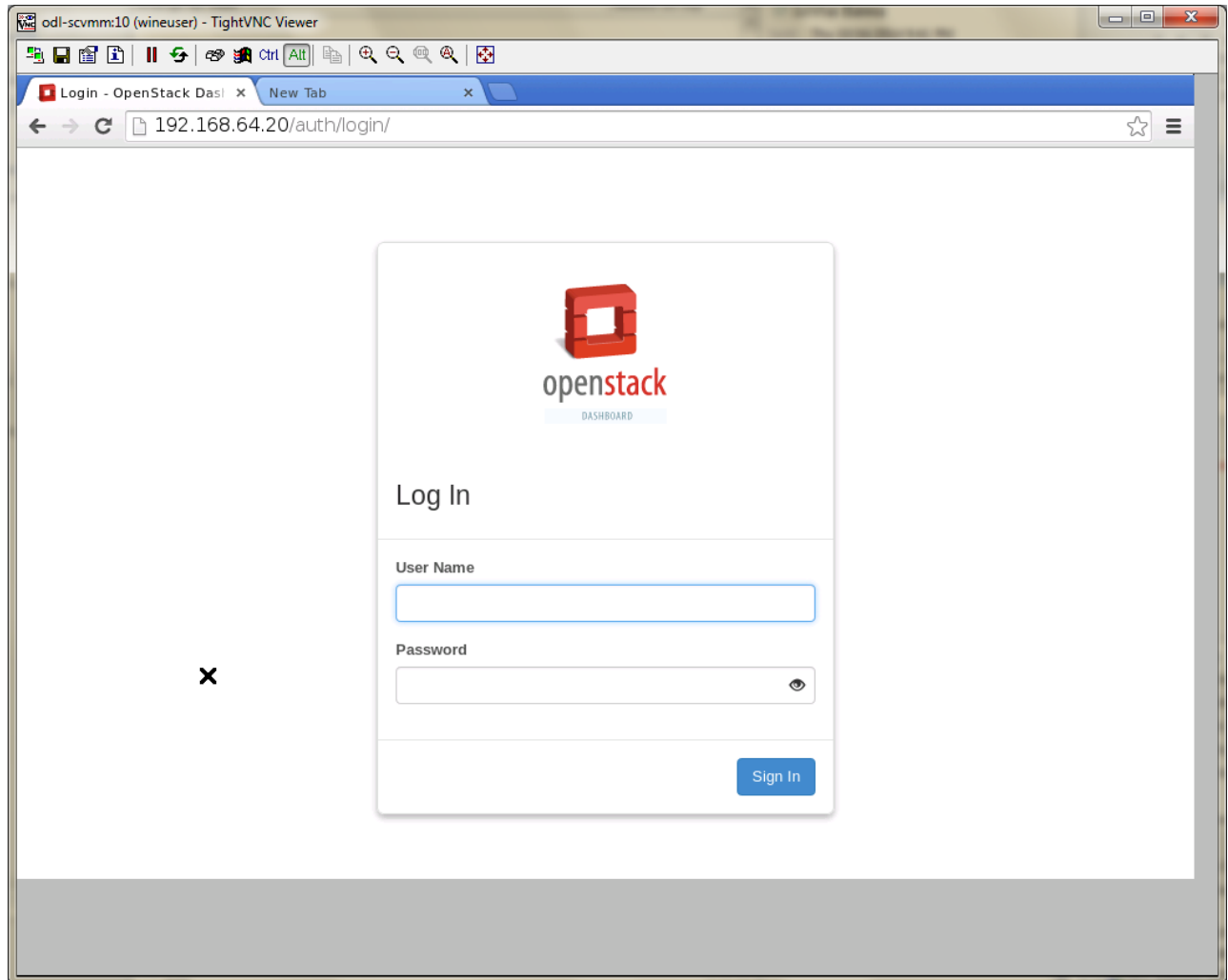
Tip

Please Accept Promiscuous mode in the networks involving the interconnect.

Create VM from Devstack Horizon GUI

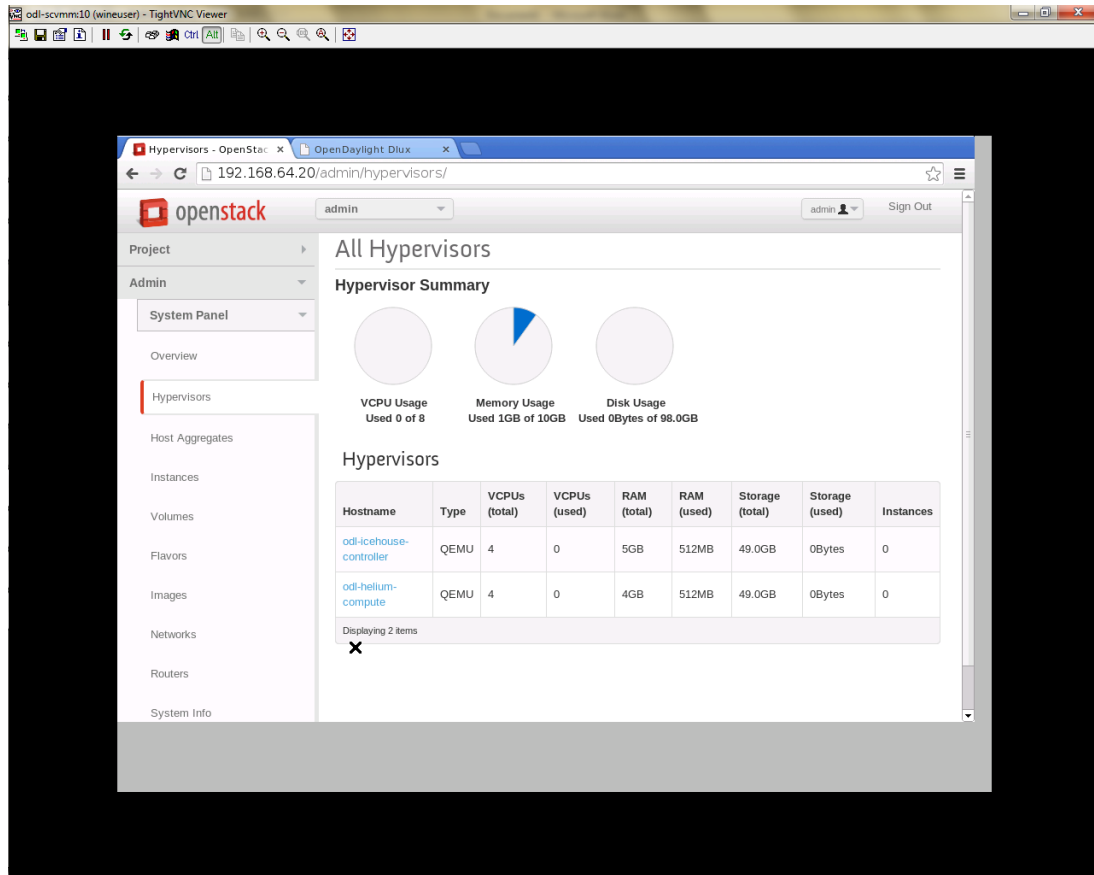
- Login to http://<CONTROL_NODE_IP>:8080/ to check the horizon GUI.

Figure 13.7. Horizon GUI



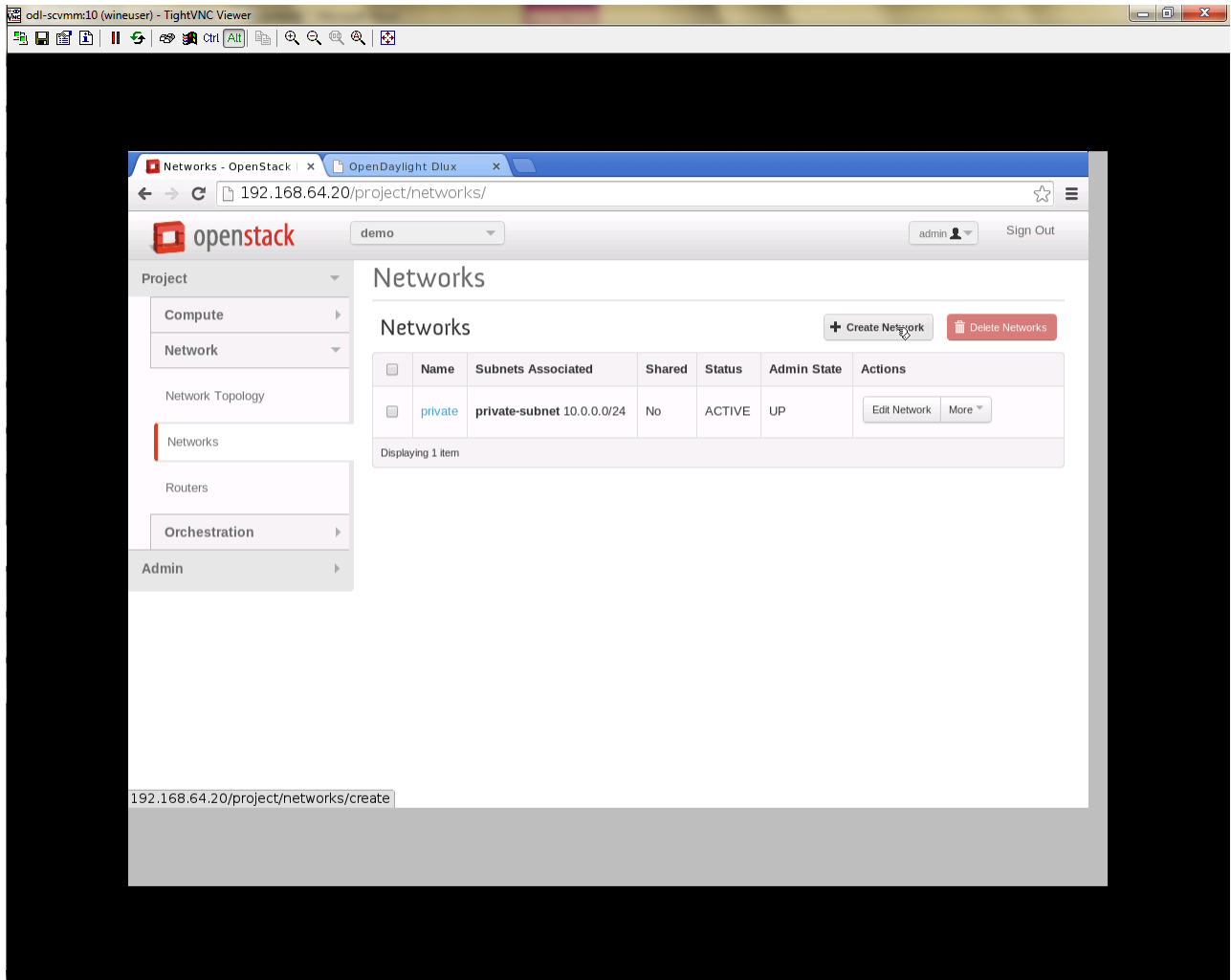
Enter the value for User Name as admin and enter the value for Password as labstack.
* We should first ensure both the hypervisors(control node and compute node) are mapped under hypervisors by clicking on Hpevisors tab.

Figure 13.8. Hypervisors



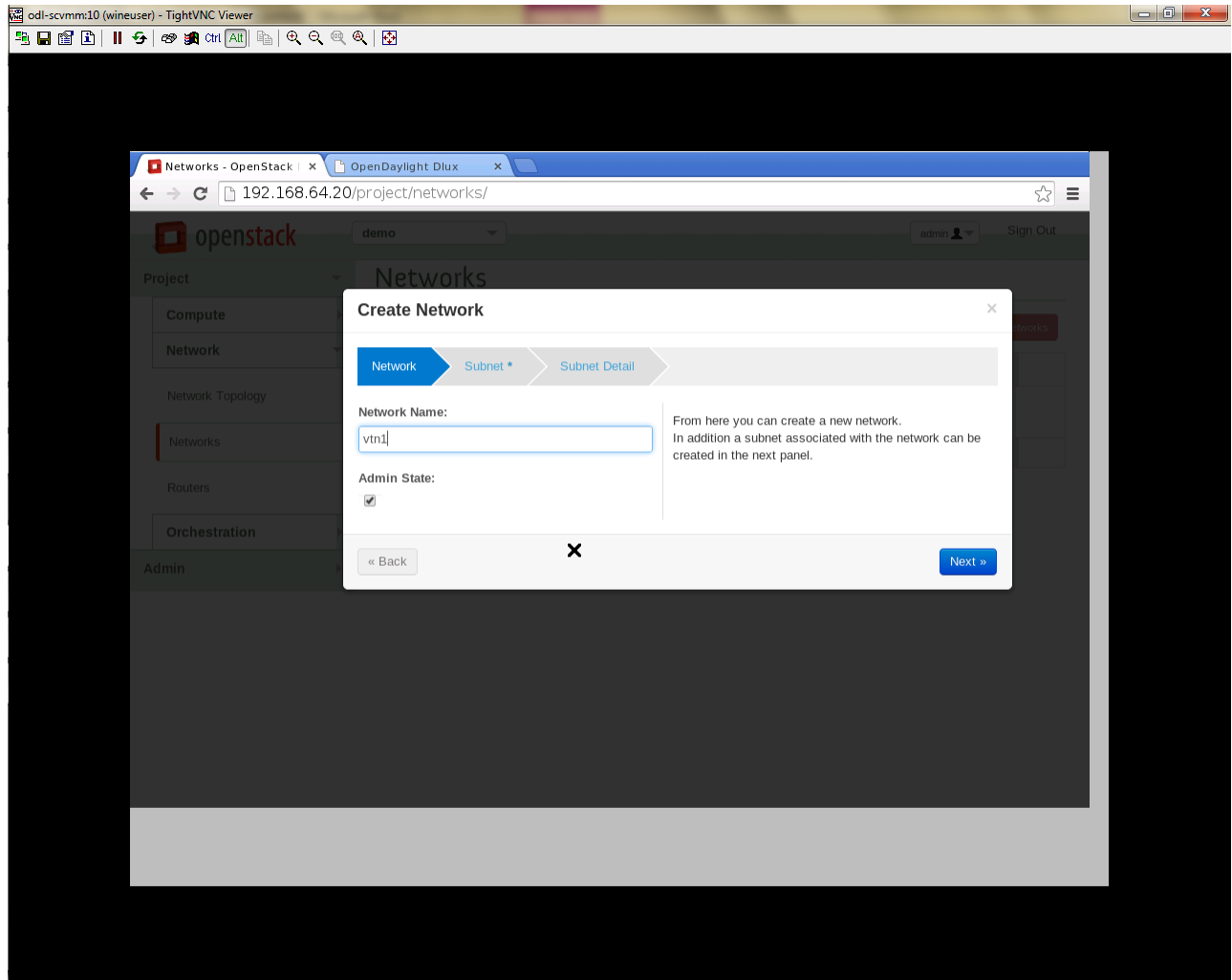
- Create a new Network from Horizon GUI.
- Click on Networks Tab.
- click on the Create Network button.

Figure 13.9. Create Network



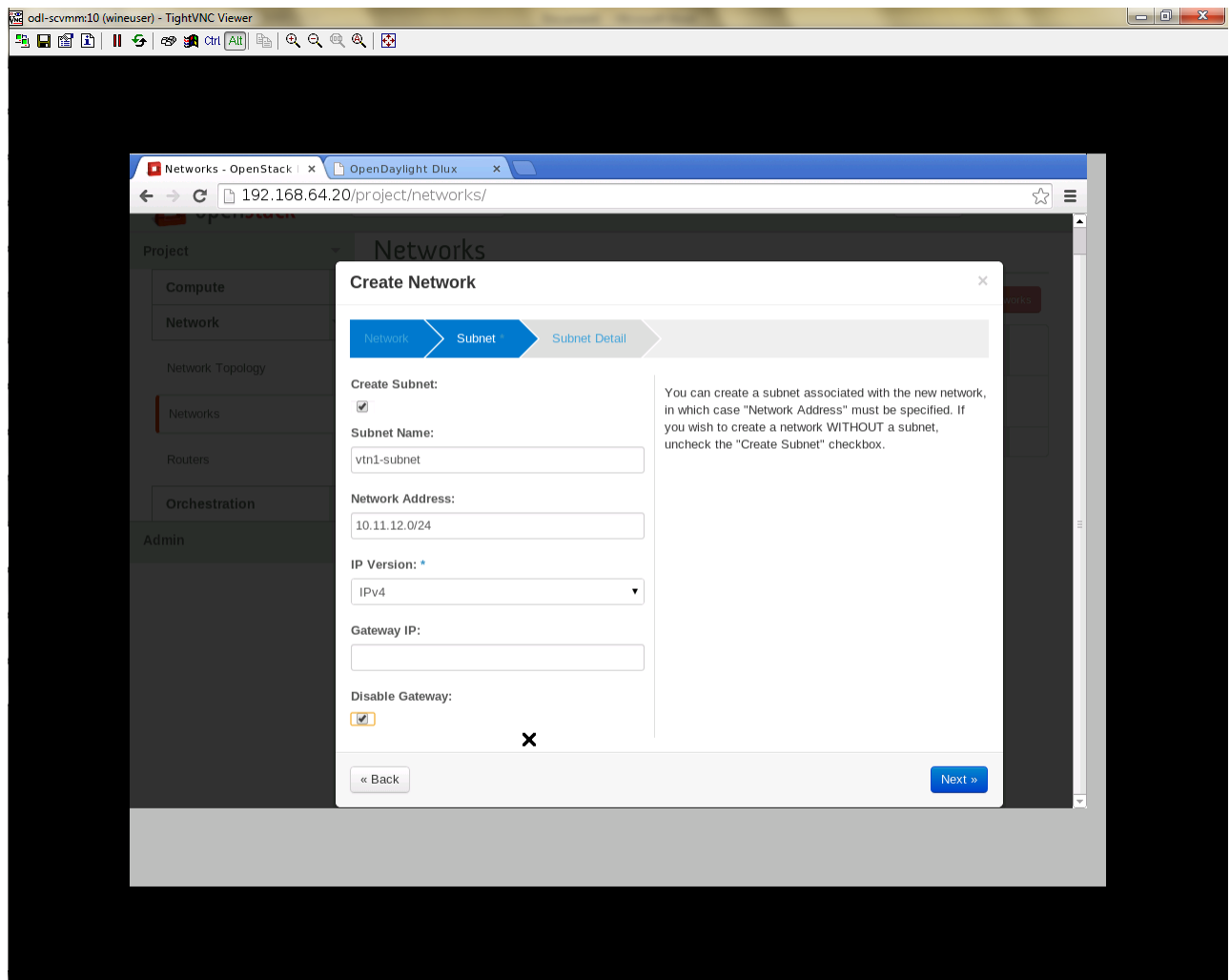
- A popup screen will appear.
- Enter network name and click Next button.

Figure 13.10. Step 1



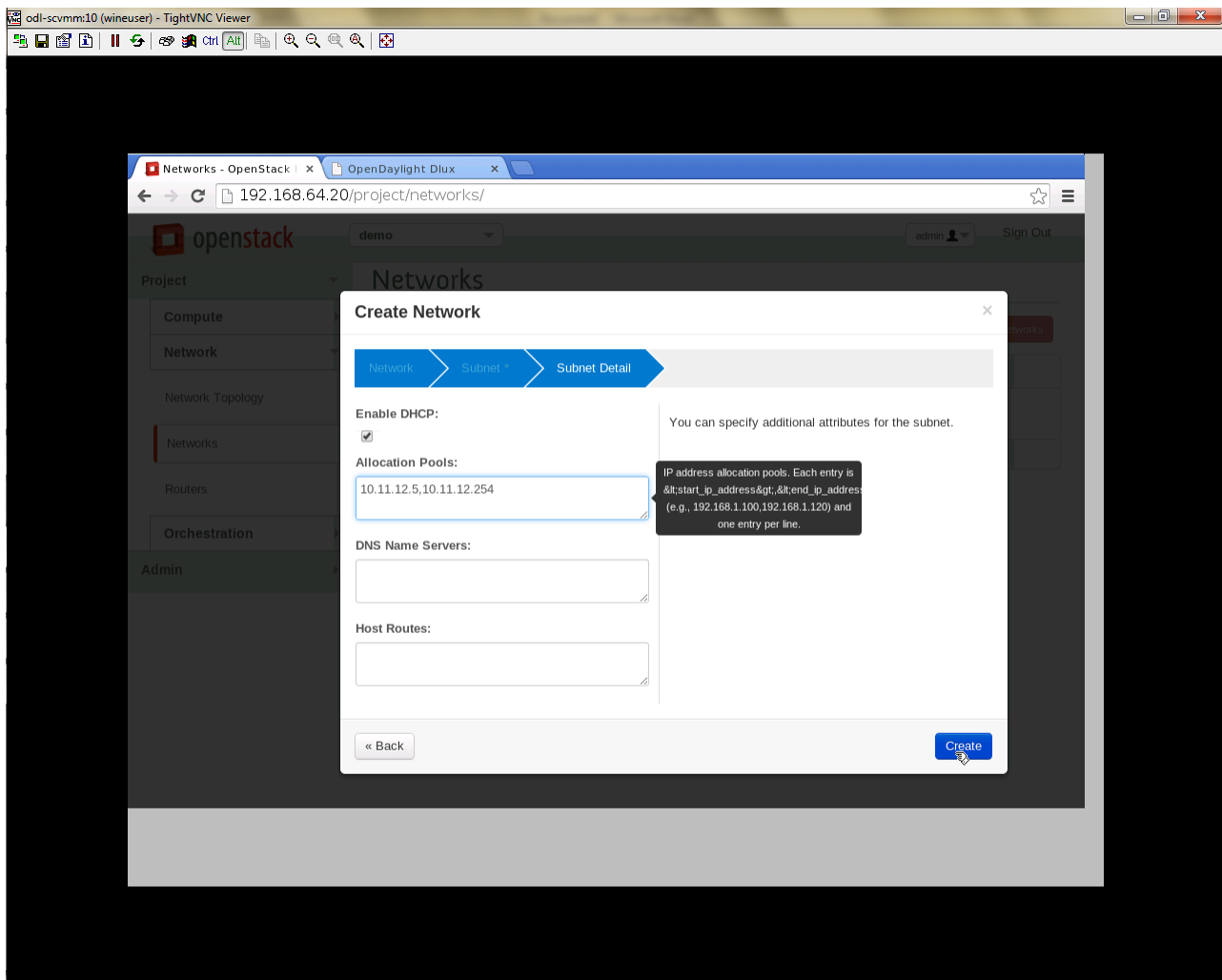
- Create a sub network by giving Network Address and click Next button .

Figure 13.11. Step 2



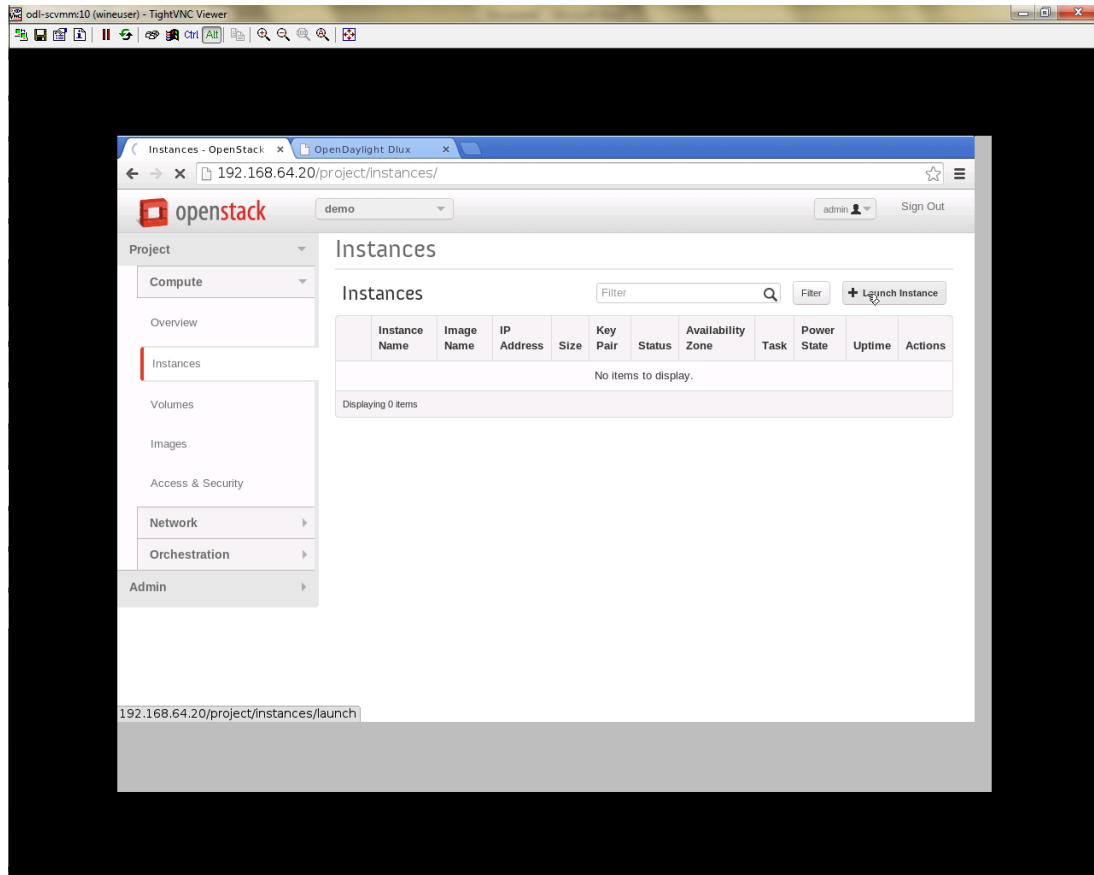
- Specify the additional details for subnetwork (please refer the image for your reference).

Figure 13.12. Step 3



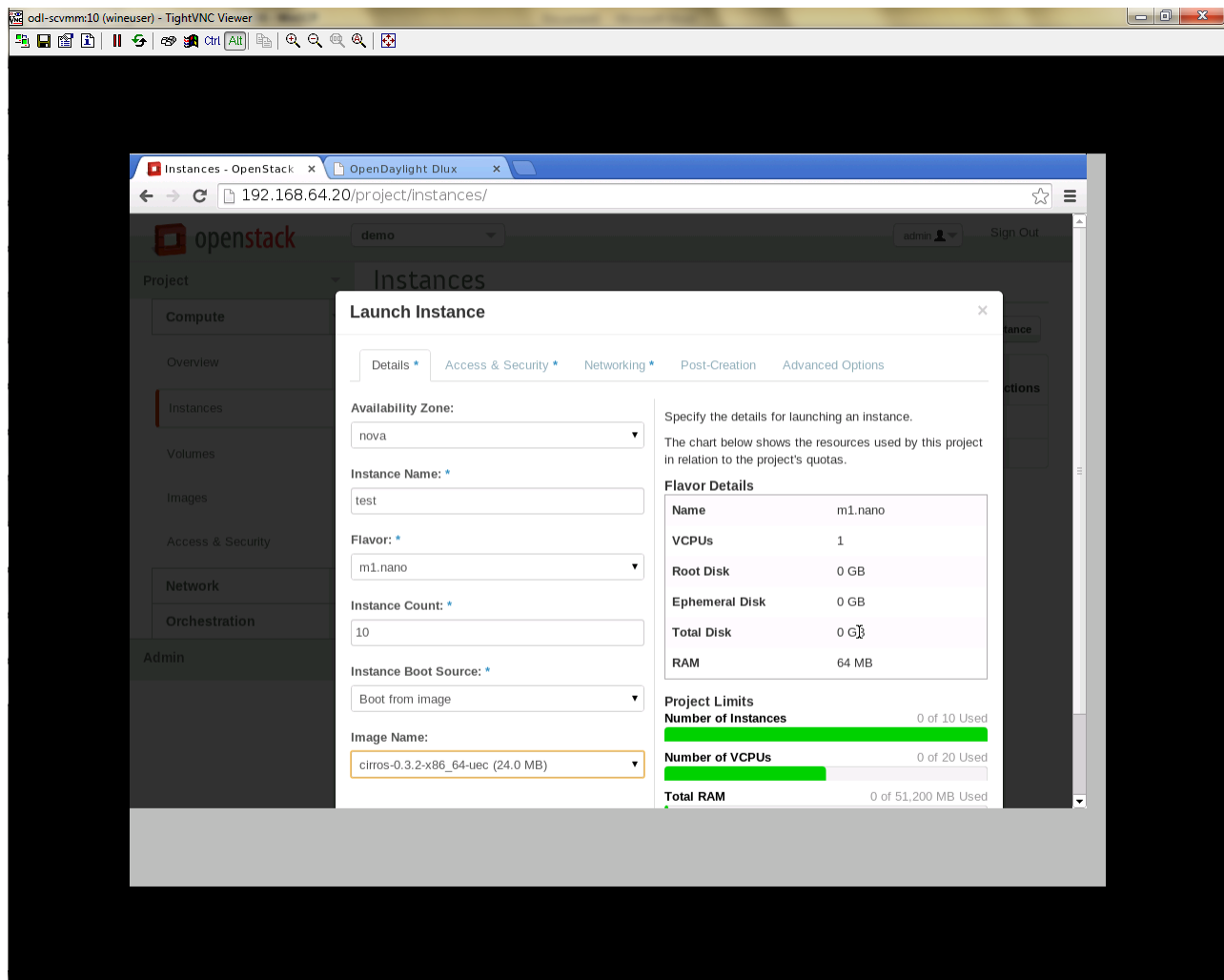
- Click Create button
- Create VM Instance
- Navigate to Instances tab in the GUI.

Figure 13.13. Instance Creation



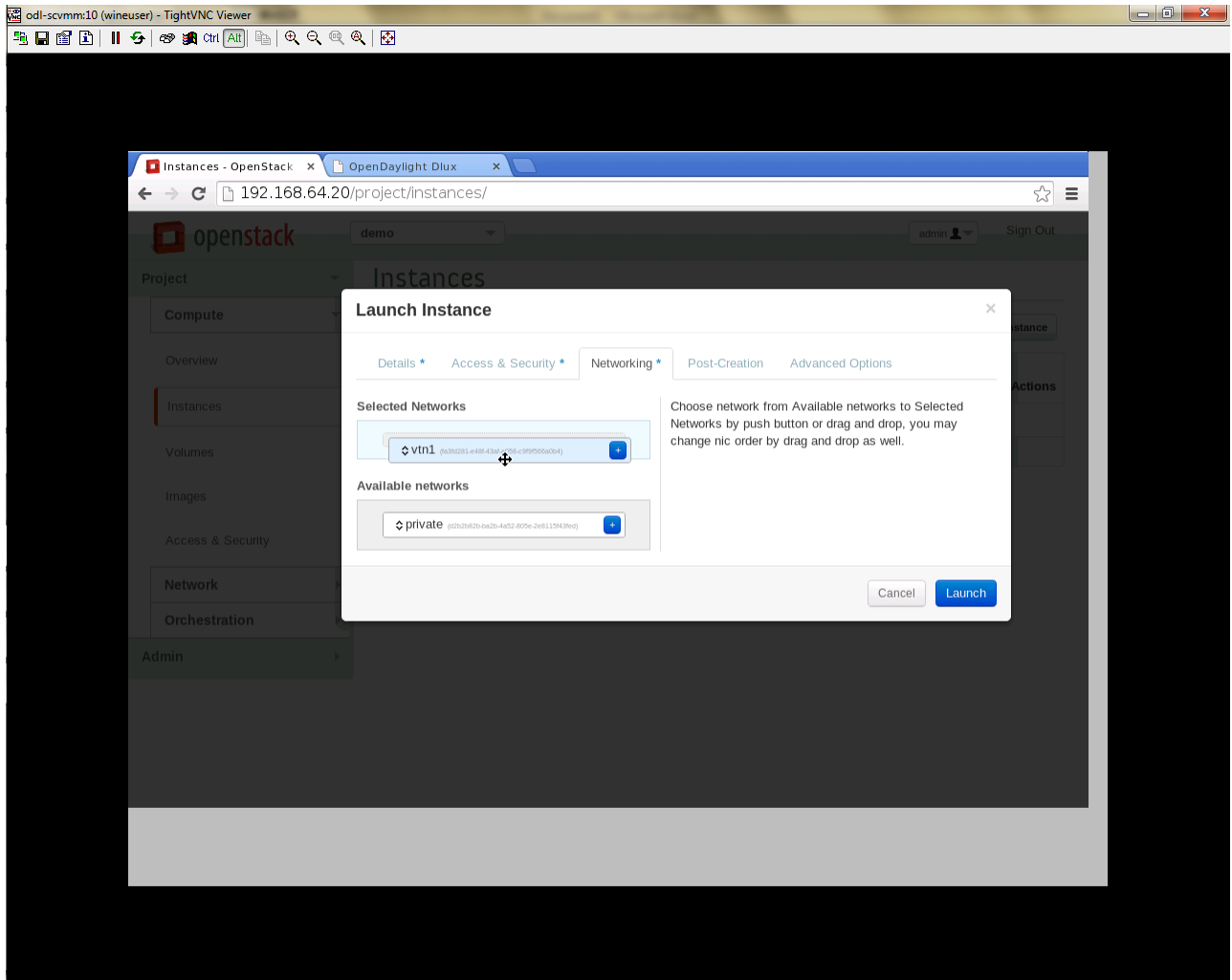
- Click on Launch Instances button.

Figure 13.14. Launch Instance



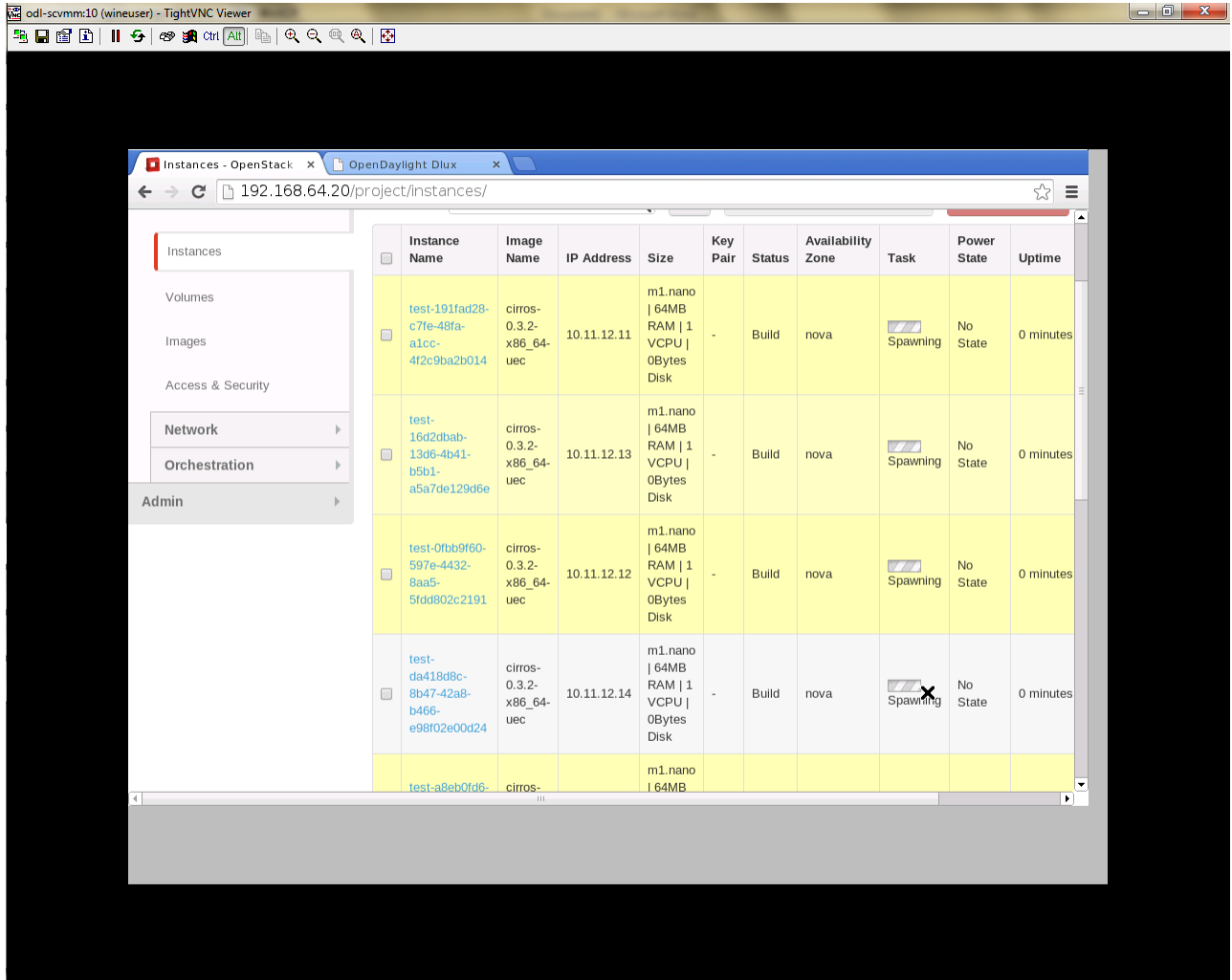
- Click on Details tab to enter the VM details. For this demo we are creating Ten VM's (instances).
- In the Networking tab, we must select the network, for this we need to drag and drop the Available networks to Selected Networks (i.e. Drag vtn1 we created from Available networks to Selected Networks and click Launch to create the instances).

Figure 13.15. Launch Network



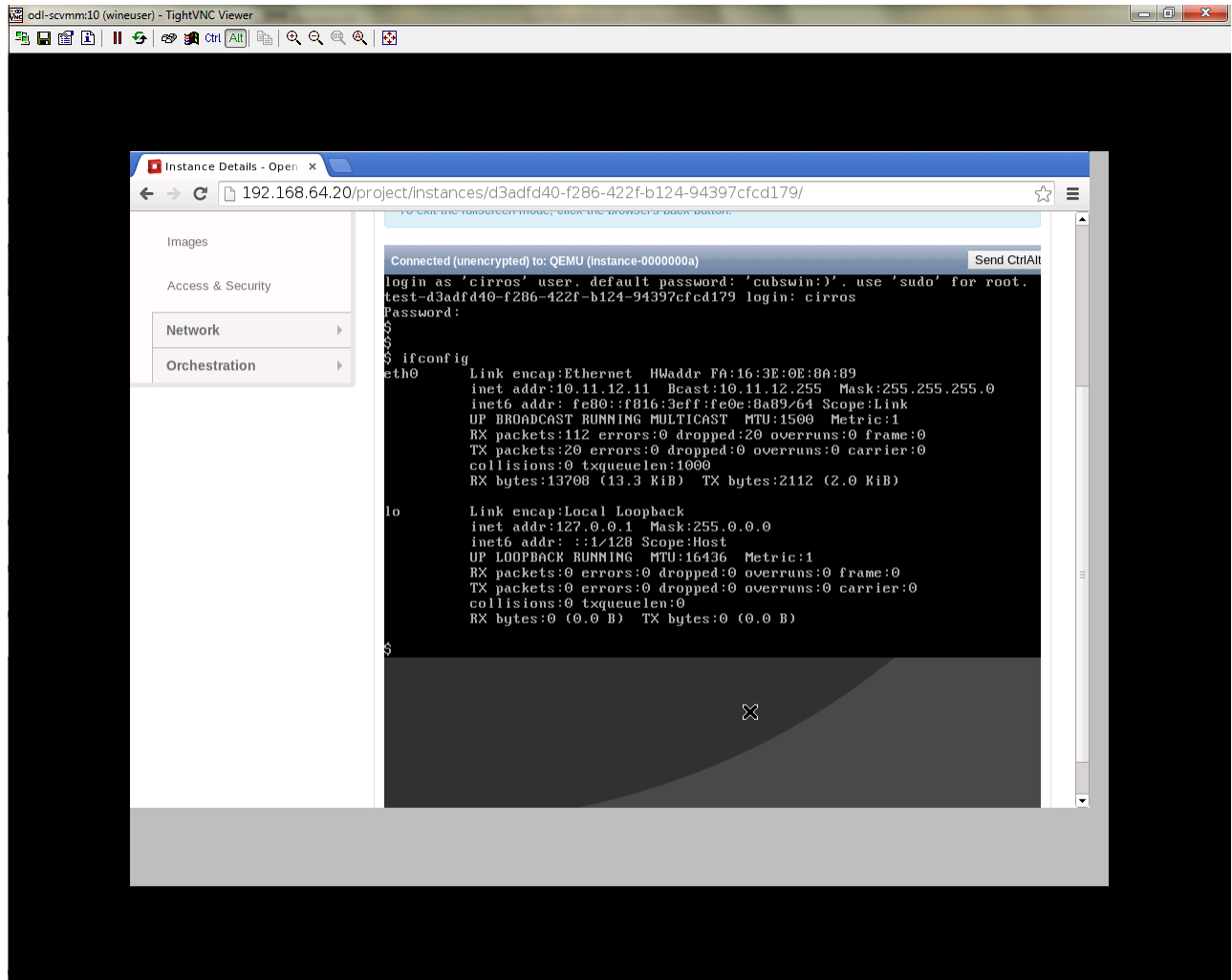
- Ten VM's will be created.

Figure 13.16. Load All Instances



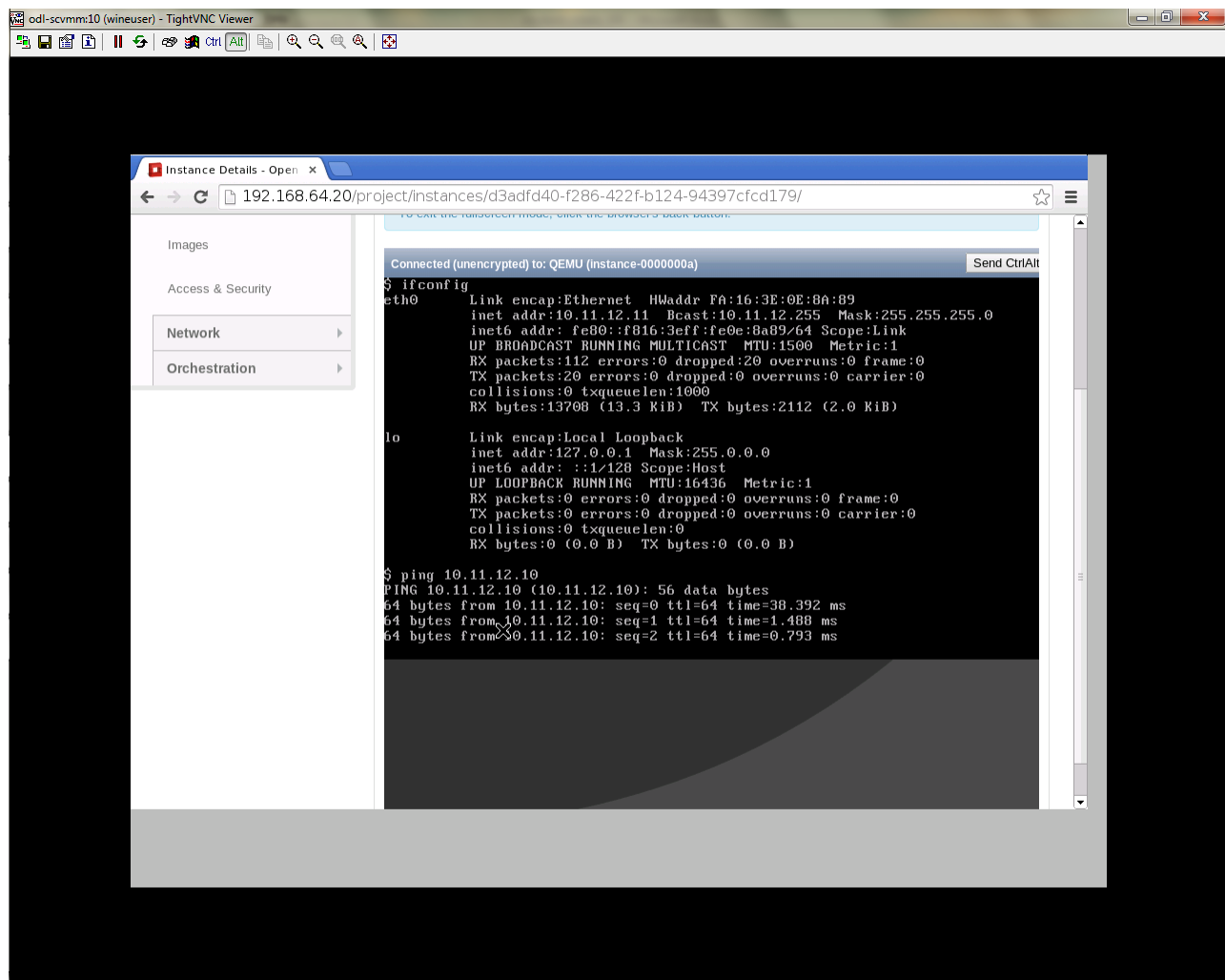
- Click on any VM displayed in the Instances tab and click the Console tab.

Figure 13.17. Instance Console



- Login to the VM console and verify with a ping command.

Figure 13.18. Ping



Verification of Control and Compute Node after VM creation

The output of `sudo ovs-vsctl` command after VM creation

```

[stack@icehouse-compute-odl devstack]$ sudo ovs-vsctl show Manager
"tcp:192.168.64.73:6640"
  is_connected: true
  Bridge br-int
  Controller "tcp:192.168.64.73:6633"
  is_connected: true
  fail_mode: secure
  Port "tapa2e1ef67-79"
  Interface "tapa2e1ef67-79"
  Port "tap5f34d39d-5e"
  Interface "tap5f34d39d-5e"
  Port "tapc2858395-f9"
  Interface "tapc2858395-f9"
  Port "tapa9ea900a-4b"
  Interface "tapa9ea900a-4b"
  Port "tapc63ef3de-53"

```

```

Interface "tapc63ef3de-53"
Port "tap01d51478-8b"
Interface "tap01d51478-8b"
Port "tapa0b085ab-ce"
Interface "tapa0b085ab-ce"
  Port "tapeab380de-8f"
Interface "tapeab380de-8f"
Port "tape404538c-0a"
Interface "tape404538c-0a"
Port "tap2940658d-15"
Interface "tap2940658d-15"
Port "ens224"
Interface "ens224"
ovs_version: "2.3.0"
<code>[stack@icehouse-controller-odl devstack]$ sudo ovs-vsctl show
Manager "tcp:192.168.64.73:6640"
  is_connected: true
  Bridge br-int
  Controller "tcp:192.168.64.73:6633"
  is_connected: true
  fail_mode: secure
  Port "tap71790d18-65"
  Interface "tap71790d18-65"
  Port "ens224"
  Interface "ens224"
  ovs_version: "2.3.0"

```

NOTE: In the above scenario more nodes have been created in the compute node

VTN Devstack Script

- The local.conf is a user-maintained settings file. This allows all custom settings for DevStack to be contained in a single file. This file is processed strictly in sequence. The following data are needed to be set in the local.conf file:
- Set the Host_IP as the detection is unreliable.
- Set FLOATING_RANGE to a range not used on the local network, i.e. 192.168.1.224/27. This configures IP addresses ending in 225-254 to be used as floating IPs.
- Set FLAT_INTERFACE to the Ethernet interface that connects the host to your local network. This is the interface that should be configured with the static IP address mentioned above.
- If the *_PASSWORD variables are not set, we will be prompted to enter values during the execution of stack.sh.
- Set ADMIN_PASSWORD . This password is used for the admin and demo accounts set up as OpenStack users. We can login to the OpenStack GUI with this credentials only.
- Set the MYSQL_PASSWORD. The default here is a random hex string which is inconvenient if you need to look at the database directly for anything.
- Set the RABBIT_PASSWORD. This is used by messaging services used by both the nodes.
- Set the service password. This is used by the OpenStack services (Nova, Glance, etc) to authenticate with Keystone.

local.conf (devstack control node)**local.conf(control)**

```

#IP Details
HOST_IP=<CONTROL_NODE_MANAGEMENT_IF_IP_ADDRESS>#Please Add The Control Node IP
Address in this line
FLAT_INTERFACE=<FLAT_INTERFACE_NAME>
SERVICE_HOST=$HOST_IP
#Instance Details
MULTI_HOST=1
#config Details
RECLONE=yes #Make it "no" after stacking successfully the first time
VERBOSE=True
LOG_COLOR=True
LOGFILE=/opt/stack/logs/stack.sh.log
SCREEN_LOGDIR=/opt/stack/logs
#OFFLINE=True #Uncomment this after stacking successfully the first time
#Passwords
ADMIN_PASSWORD=labstack
MYSQL_PASSWORD=supersecret
RABBIT_PASSWORD=supersecret
SERVICE_PASSWORD=supersecret
SERVICE_TOKEN=supersecrettoken
ENABLE_TENANT_TUNNELS=false
#Services
disable_service rabbit
enable_service qpid
enable_service quantum
enable_service n-cpu
enable_service n-cond
disable_service n-net
enable_service q-svc
enable_service q-dhcp
enable_service q-meta
enable_service horizon
enable_service quantum
enable_service tempest
ENABLED_SERVICES+=,n-api,n-crt,n-obj,n-cpu,n-cond,n-sch,n-novnc,n-cauth,n-
cauth,nova
ENABLED_SERVICES+=,cinder,c-api,c-vol,c-sch,c-bak
#ML2 Details
Q_PLUGIN=ml2
Q_ML2_PLUGIN_MECHANISM_DRIVERS=opendaylight
Q_ML2_TENANT_NETWORK_TYPE=local
Q_ML2_PLUGIN_TYPE_DRIVERS=local
disable_service n-net
enable_service q-svc
enable_service q-dhcp
enable_service q-meta
enable_service neutron
enable_service odl-compute
ODL_MGR_IP=<ODL_IP_ADDRESS> #Please Add the ODL IP Address in this line
OVS_PHYSICAL_BRIDGE=br-int
Q_OVS_USE_VETH=True
url=http://<ODL_IP_ADDRESS>:8080/controller/nb/v2/neutron #Please Add the ODL
IP Address in this line
username=admin
password=admin

```

local.conf (devstack compute node)

local.conf(compute)

```

#IP Details
HOST_IP=<COMPUTE_NODE_MANAGEMENT_IP_ADDRESS> #Add the Compute node Management
IP Address
SERVICE_HOST=<CONTROLLER_NODE_MANAGEMENT_IP_ADDRESS> #Add the cotnrol Node
Management IP Address here
#Instance Details
MULTI_HOST=1
#config Details
RECLONE=yes #Make thgis "no" after stacking successfully once
#OFFLINE=True #Uncomment this line after stacking successfully first time.
VERBOSE=True
LOG_COLOR=True
LOGFILE=/opt/stack/logs/stack.sh.log
SCREEN_LOGDIR=/opt/stack/logs
#Passwords
ADMIN_PASSWORD=labstack
MYSQL_PASSWORD=supersecret
RABBIT_PASSWORD=supersecret
SERVICE_PASSWORD=supersecret
SERVICE_TOKEN=supersecrettoken
#Services
ENABLED_SERVICES=n-cpu,rabbit,neutron
#ML2 Details
Q_PLUGIN=ml2
Q_ML2_PLUGIN_MECHANISM_DRIVERS=opendaylight
Q_ML2_TENANT_NETWORK_TYPE=local
Q_ML2_PLUGIN_TYPE_DRIVERS=local
enable_service odl-compute
ODL_MGR_IP=<ODL_IP_ADDRESS> #ADD ODL IP address here
OVS_PHYSICAL_BRIDGE=br-int
ENABLE_TENANT_TUNNELS=false
Q_OVS_USE_VETH=True
#Details of the Control node for various services
[[post-config|/etc/neutron/plugins/ml2/ml2_conf.ini]]
Q_HOST=$SERVICE_HOST
MYSQL_HOST=$SERVICE_HOST
RABBIT_HOST=$SERVICE_HOST
GLANCE_HOSTPORT=$SERVICE_HOST:9292
KEYSTONE_AUTH_HOST=$SERVICE_HOST
KEYSTONE_SERVICE_HOST=$SERVICE_HOST
NOVA_VNC_ENABLED=True
NOVNCPROXY_URL="http://<CONTROLLER_NODE_IP_ADDRESS>:6080/vnc_auto.html" #Add
Controller Node IP address
VNCSERVER_LISTEN=$HOST_IP
VNCSERVER_PROXYCLIENT_ADDRESS=$VNCSERVER_LISTEN

```



Note

We have to comment OFFLINE=TRUE in local.conf files, this will make all the installations to happen automatically. RECLONE=yes only when we set up the DevStack environment from scratch.

References

- <http://devstack.org/guides/multinode-lab.html>

- https://wiki.opendaylight.org/view/File:Vtn_demo_hackfest_2014_march.pdf

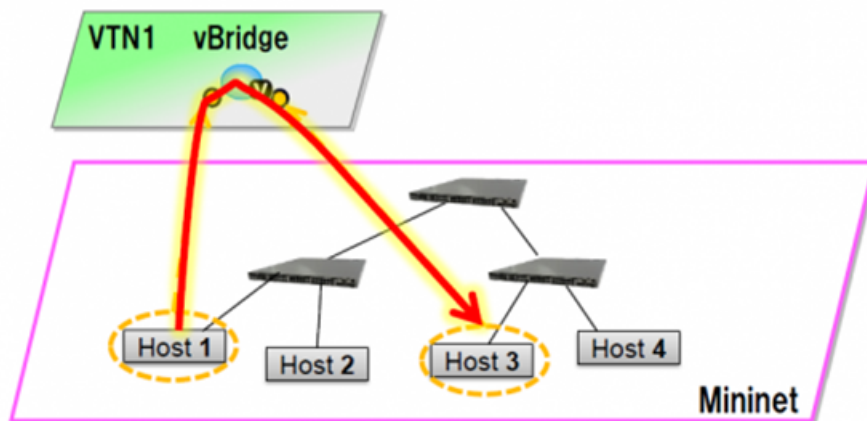
VTN Usage Examples

How to configure L2 Network with Single Controller

Overview

This example provides the procedure to demonstrate configuration of VTN Coordinator with L2 network using VTN Virtualization(single controller). Here is the Example for vBridge Interface Mapping with Single Controller using mininet. mininet details and set-up can be referred at below URL: https://wiki.opendaylight.org/view/OpenDaylight_Controller:Installation#Using_Mininet

Figure 13.19. EXAMPLE DEMONSTRATING SINGLE CONTROLLER



Requirements

- Configure mininet and create a topology:

```
mininet@mininet-vm:~$ sudo mn --controller=remote,ip=<controller-ip> --topo
tree,2
```

- mininet> net

```
s1 lo: s1-eth1:h1-eth0 s1-eth2:s2-eth1
s2 lo: s2-eth1:s1-eth2 s2-eth2:h2-eth0
h1 h1-eth0:s1-eth1
h2 h2-eth0:s2-eth2
```

Configuration

- Create a Controller

```
curl --user admin:adminpass -H 'content-type: application/json' -X POST -
d '{"controller": {"controller_id": "controllerone", "ipaddr": "10.0.0.2",
```

```
"type": "odc", "version": "1.0", "auditstatus": "enable"}}' http://127.0.0.1:8083/vtn-webapi/controllers.json
```

- Create a VTN

```
curl --user admin:adminpass -H 'content-type: application/json' -X POST -d '{"vtn" : {"vtn_name": "vtn1", "description": "test VTN" }}' http://127.0.0.1:8083/vtn-webapi/vtns.json
```

- Create a vBridge in the VTN

```
curl --user admin:adminpass -H 'content-type: application/json' -X POST -d '{"vbridge" : {"vbr_name": "vBridge1", "controller_id": "controllerone", "domain_id": "(DEFAULT)" }}' http://127.0.0.1:8083/vtn-webapi/vtns/vtn1/vbridges.json
```

- Create two Interfaces into the vBridge

```
curl --user admin:adminpass -H 'content-type: application/json' -X POST -d '{"interface": {"if_name": "if1", "description": "if_desc1" }}' http://127.0.0.1:8083/vtn-webapi/vtns/vtn1/vbridges/vBridge1/interfaces.json
```

```
curl --user admin:adminpass -H 'content-type: application/json' -X POST -d '{"interface": {"if_name": "if2", "description": "if_desc2" }}' http://127.0.0.1:8083/vtn-webapi/vtns/vtn1/vbridges/vBridge1/interfaces.json
```

- Get the list of logical ports configured

```
Curl --user admin:adminpass -H 'content-type: application/json' -X GET http://127.0.0.1:8083/vtn-webapi/controllers/controllerone/domains/(DEFAULT)/logical_ports.json
```

- Configure two mappings on the interfaces

```
curl --user admin:adminpass -H 'content-type: application/json' -X PUT -d '{"portmap": {"logical_port_id": "PP-OF:00:00:00:00:00:00:00:03-s3-eth1" }}' http://127.0.0.1:8083/vtn-webapi/vtns/vtn1/vbridges/vBridge1/interfaces/if1/portmap.json
curl --user admin:adminpass -H 'content-type: application/json' -X PUT -d '{"portmap": {"logical_port_id": "PP-OF:00:00:00:00:00:00:00:02-s2-eth1" }}' http://127.0.0.1:8083/vtn-webapi/vtns/vtn1/vbridges/vBridge1/interfaces/if2/portmap.json
```

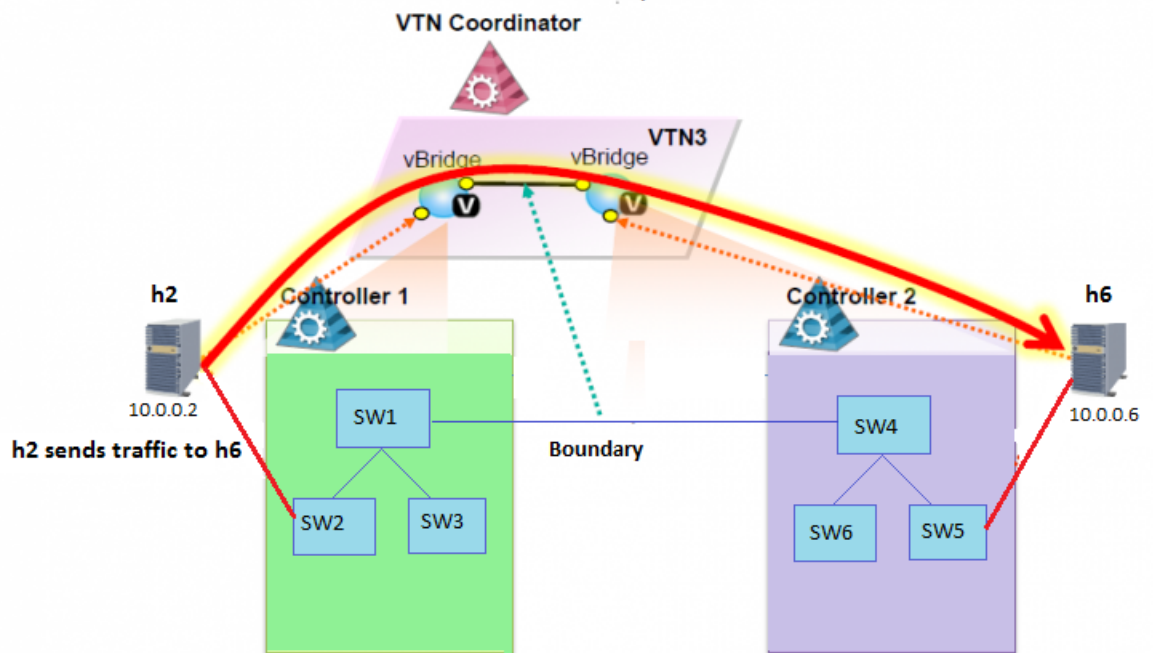
Verification

Please verify whether the Host1 and Host3 are pinging. * Send packets from Host1 to Host3

```
|mininet> h1 ping h3
```

How to configure L2 Network with Multiple Controllers

- This example provides the procedure to demonstrate configuration of VTN Coordinator with L2 network using VTN Virtualization Here is the Example for vBridge Interface Mapping with Multi-controller using mininet.

Figure 13.20. EXAMPLE DEMONSTRATING MULTIPLE CONTROLLERS

Requirements

- Configure multiple controllers using the mininet script given below: https://wiki.opendaylight.org/view/OpenDaylight_Virtual_Tenant_Network_%28VTN%29:Scripts:Mininet#Network_with_Multiple_Paths_for_delivering_packets

Configuration

- Create a VTN

```
curl --user admin:adminpass -H 'content-type: application/json' -X POST -d '{"vtn" : {"vtn_name": "vtn3"}}' http://127.0.0.1:8083/vtn-webapi/vtns.json
```

- Create two Controllers

```
curl --user admin:adminpass -H 'content-type: application/json' -X POST -d '{"controller": {"controller_id": "odc1", "ipaddr": "10.100.9.52", "type": "odc", "version": "1.0", "auditstatus": "enable"}}' http://127.0.0.1:8083/vtn-webapi/controllers.json
```

```
curl --user admin:adminpass -H 'content-type: application/json' -X POST -d '{"controller": {"controller_id": "odc2", "ipaddr": "10.100.9.61", "type": "odc", "version": "1.0", "auditstatus": "enable"}}' http://127.0.0.1:8083/vtn-webapi/controllers.json
```

- Create two vBridges in the VTN like, vBridge1 in Controller1 and vBridge2 in Controller2

```
curl --user admin:adminpass -H 'content-type: application/json' -X POST -d '{"vbridge" : {"vbr_name": "vbr1", "controller_id": "odc1", "domain_id": "(DEFAULT)" }}' http://127.0.0.1:8083/vtn-webapi/vtns/vtn3/vbridges.json
```

```
curl --user admin:adminpass -H 'content-type: application/json' -X POST -d '{"vbridge" : {"vbr_name": "vbr2", "controller_id": "odc2", "domain_id": "(DEFAULT)" }}' http://127.0.0.1:8083/vtn-webapi/vtns/vtn3/vbridges.json
```

- Create vBridge Interfaces

```
curl --user admin:adminpass -H 'content-type: application/json' -X POST -d '{"interface": {"if_name": "if1"}}' http://127.0.0.1:8083/vtn-webapi/vtns/vtn3/vbridges/vbr1/interfaces.json
```

```
curl --user admin:adminpass -H 'content-type: application/json' -X POST -d '{"interface": {"if_name": "if2"}}' http://127.0.0.1:8083/vtn-webapi/vtns/vtn3/vbridges/vbr1/interfaces.json
```

```
curl --user admin:adminpass -H 'content-type: application/json' -X POST -d '{"interface": {"if_name": "if1"}}' http://127.0.0.1:8083/vtn-webapi/vtns/vtn3/vbridges/vbr2/interfaces.json
```

```
curl --user admin:adminpass -H 'content-type: application/json' -X POST -d '{"interface": {"if_name": "if2"}}' http://127.0.0.1:8083/vtn-webapi/vtns/vtn3/vbridges/vbr2/interfaces.json
```

- Get the list of logical ports configured

```
curl --user admin:adminpass -H 'content-type: application/json' -X GET http://127.0.0.1:8083/vtn-webapi/controllers/odc1/domains/(DEFAULT)/logical_ports/detail.json
```

- Create boundary and vLink

```
curl --user admin:adminpass -H 'content-type: application/json' -X POST -d '{"boundary": {"boundary_id": "b1", "link": {"controller1_id": "odc1", "domain1_id": "(DEFAULT)", "logical_port1_id": "PP-OF:00:00:00:00:00:00:00:01-s1-eth3", "controller2_id": "odc2", "domain2_id": "(DEFAULT)", "logical_port2_id": "PP-OF:00:00:00:00:00:00:00:04-s4-eth3"}}}' http://127.0.0.1:8083/vtn-webapi/boundaries.json
```

```
curl --user admin:adminpass -H 'content-type: application/json' -X POST -d '{"vlink": {"vlnk_name": "vlink1", "vnode1_name": "vbr1", "if1_name": "if2", "vnode2_name": "vbr2", "if2_name": "if2", "boundary_map": {"boundary_id": "b1", "vlan_id": "50"}}}' http://127.0.0.1:8083/vtn-webapi/vtns/vtn3/vlinks.json
```

- Configure port-map on the interfaces

```
curl --user admin:adminpass -H 'content-type: application/json' -X PUT -d '{"portmap": {"logical_port_id": "PP-OF:00:00:00:00:00:00:00:02-s2-eth2"}}' http://127.0.0.1:8083/vtn-webapi/vtns/vtn3/vbridges/vbr1/interfaces/if1/portmap.json
```

```
curl --user admin:adminpass -H 'content-type: application/json' -X PUT -d '{"portmap": {"logical_port_id": "PP-OF:00:00:00:00:00:00:00:05-s5-eth2"}}' http://127.0.0.1:8083/vtn-webapi/vtns/vtn3/vbridges/vbr2/interfaces/if1/portmap.json
```

Verification

Please verify whether Host h2 and Host h6 are pinging. * Send packets from h2 to h6

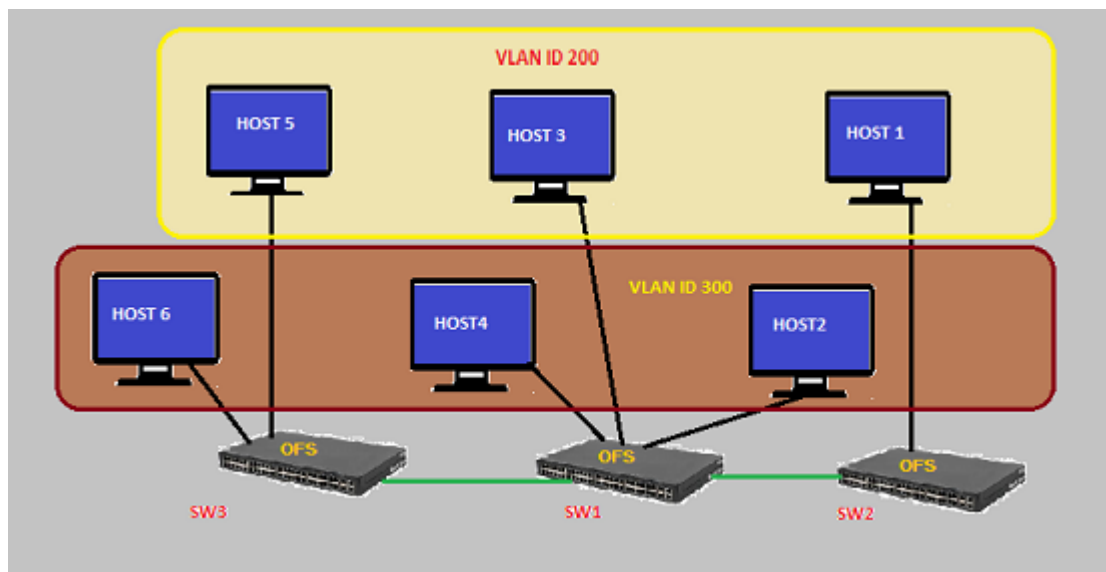
```
mininet> h2 ping h6
```

How To Test Vlan-Map In Mininet Environment

Overview

This example explains how to test vlan-map in a multi host scenario.

Figure 13.21. Example that demonstrates vlanmap testing in Mininet Environment



Requirements

- Save the mininet script given below as `vlan_vtn_test.py` and run the mininet script in the mininet environment where Mininet is installed.

Mininet Script

[https://wiki.opendaylight.org/view/OpenDaylight_Virtual_Tenant_Network_\(VTN\):Scripts:Mininet#Network_with_hosts_in_different_vlan](https://wiki.opendaylight.org/view/OpenDaylight_Virtual_Tenant_Network_(VTN):Scripts:Mininet#Network_with_hosts_in_different_vlan)

- Run the mininet script

```
sudo mn --controller=remote,ip=192.168.64.13 --custom vlan_vtn_test.py --topo mytopo
```

Configuration

Please follow the below steps to test a vlan map using mininet: * Create a controller

```
curl --user admin:adminpass -H 'content-type: application/json' -X POST -d '{"controller": {"controller_id": "controllerone", "ipaddr": "10.0.0.2", "type": "odc", "version": "1.0", "auditstatus": "enable"}}' http://127.0.0.1:8083/vtn-webapi/controllers
```

- Create a VTN

```
curl -X POST -H 'content-type: application/json' -H 'username: admin' -H
'password: adminpass' -d '{"vtn" : {"vtn_name": "vtn1", "description": "test
VTN" }}' http://127.0.0.1:8083/vtn-webapi/vtns.json
```

- Create a vBridge(vBridge1)

```
curl -X POST -H 'content-type: application/json' -H 'username: admin'
-H 'password: adminpass' -d '{"vbridge" : {"vbr_name": "vBridge1",
"controller_id": "controllerone", "domain_id": "(DEFAULT)" }}' http://127.0.0.
1:8083/vtn-webapi/vtns/vtn1/vbridges.json
```

- Create a vlan map with vlanid 200 for vBridge vBridge1

```
curl -X POST -H 'content-type: application/json' -H 'username: admin' -H
'password: adminpass' -d '{"vlanmap" : {"vlan_id": 200 }}' http://127.0.0.
1:8083/vtn-webapi/vtns/vtn1/vbridges/vBridge1/vlanmaps.json
```

- Create a vBridge (vBridge2)

```
curl -X POST -H 'content-type: application/json' -H 'username: admin'
-H 'password: adminpass' -d '{"vbridge" : {"vbr_name": "vBridge2",
"controller_id": "controllerone", "domain_id": "(DEFAULT)" }}' http://127.0.0.
1:8083/vtn-webapi/vtns/vtn1/vbridges.json
```

- Create a vlan map with vlanid 300 for vBridge vBridge2

```
curl -X POST -H 'content-type: application/json' -H 'username: admin' -H
'password: adminpass' -d '{"vlanmap" : {"vlan_id": 300 }}' http://127.0.0.
1:8083/vtn-webapi/vtns/vtn1/vbridges/vBridge2/vlanmaps.json
```

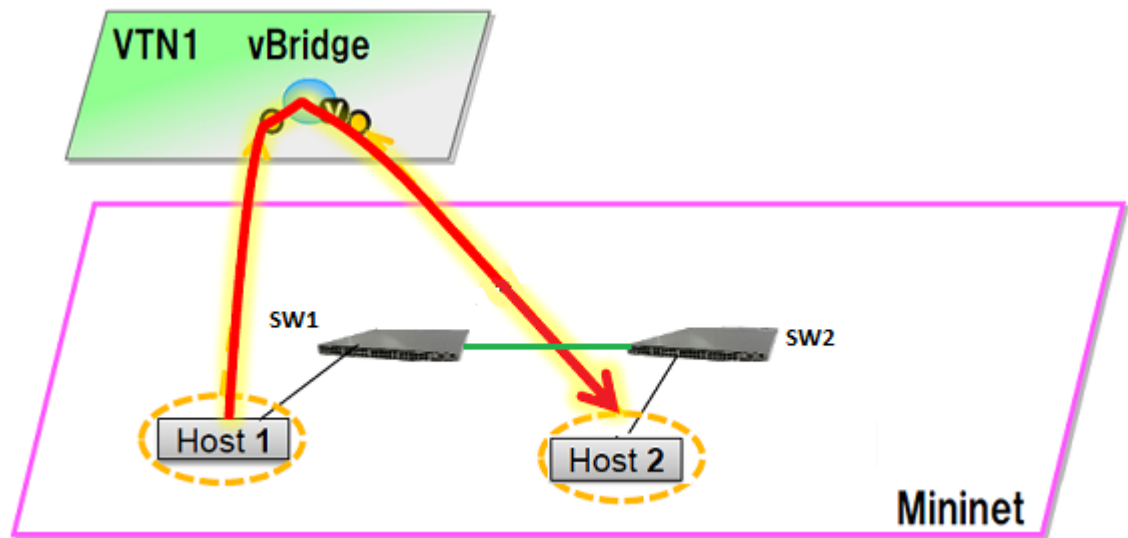
Verification

Ping all in mininet environment to view the host reachability.

```
mininet> pingall
Ping: testing ping reachability
h1 -> X h3 X h5 X
h2 -> X X h4 X h6
h3 -> h1 X X h5 X
h4 -> X h2 X X h6
h5 -> h1 X h3 X X
h6 -> X h2 X h4 X
```

How To View Specific VTN Station Information.

This example demonstrates on how to view a specific VTN Station information.

Figure 13.22. EXAMPLE DEMONSTRATING VTN STATIONS

Requirement

- Configure mininet and create a topology:

```
$ sudo mn --custom /home/mininet/mininet/custom/topo-2sw-2host.py --
controller=remote,ip=10.100.9.61 --topo mytopo
mininet> net

s1 lo: s1-eth1:h1-eth0 s1-eth2:s2-eth1
s2 lo: s2-eth1:s1-eth2 s2-eth2:h2-eth0
h1 h1-eth0:s1-eth1
h2 h2-eth0:s2-eth2
```

*Generate traffic by pinging between hosts h1 and h2 after configuring the portmaps respectively

```
mininet> h1 ping h2
PING 10.0.0.2 (10.0.0.2) 56(84) bytes of data.
64 bytes from 10.0.0.2: icmp_req=1 ttl=64 time=16.7 ms
64 bytes from 10.0.0.2: icmp_req=2 ttl=64 time=13.2 ms
```

Configuration

Create Controller.

```
curl -v --user admin:adminpass -H 'content-type: application/json' -X POST -
d '{"controller": {"controller_id": "controllerone", "ipaddr": "10.100.9.61",
"type": "odc", "version": "1.0", "auditstatus": "enable"}}' http://127.0.0.
1:8083/vtn-webapi/controllers.json
```

Create a VTN.

```
curl -v --user admin:adminpass -H 'content-type: application/json' -X POST -
d '{"vtn" : {"vtn_name": "vtn1", "description": "test VTN" }}' http://127.0.0.
1:8083/vtn-webapi/vtns.json
```

Create a vBridge in the VTN.

```
curl -v --user admin:adminpass -H 'content-type: application/json' -X POST -d '{"vbridge" : {"vbr_name": "vBridge1", "controller_id": "controllerone", "domain_id": "(DEFAULT)" }}' http://127.0.0.1:8083/vtn-webapi/vtns/vtn1/vbridges.json
```

Create two Interfaces into the vBridge.

```
curl -v --user admin:adminpass -H 'content-type: application/json' -X POST -d '{"interface": {"if_name": "if1", "description": "if_desc1"}}' http://127.0.0.1:8083/vtn-webapi/vtns/vtn1/vbridges/vBridge1/interfaces.json
curl -v --user admin:adminpass -H 'content-type: application/json' -X POST -d '{"interface": {"if_name": "if2", "description": "if_desc2"}}' http://127.0.0.1:8083/vtn-webapi/vtns/vtn1/vbridges/vBridge1/interfaces.json
```

Configure two mappings on the interfaces.

```
curl -v --user admin:adminpass -H 'content-type: application/json' -X PUT -d '{"portmap":{"logical_port_id": "PP-OF:00:00:00:00:00:00:01-s1-eth1"}}' http://127.0.0.1:8083/vtn-webapi/vtns/vtn1/vbridges/vBridge1/interfaces/if1/portmap.json
curl -v --user admin:adminpass -H 'content-type: application/json' -X PUT -d '{"portmap":{"logical_port_id": "PP-OF:00:00:00:00:00:00:02-s2-eth2"}}' http://17.0.0.1:8083/vtn-webapi/vtns/vtn1/vbridges/vBridge1/interfaces/if2/portmap.json
```

Get the VTN stations information.

```
curl -v -X GET -H 'content-type: application/json' -H 'username: admin' -H 'password: adminpass' "http://127.0.0.1:8083/vtn-webapi/vtnstations?controller_id=controllerone&vtn_name=vtn1"
```

Verification

```
curl -v -X GET -H 'content-type: application/json' -H 'username: admin' -H 'password: adminpass' "http://127.0.0.1:8083/vtn-webapi/vtnstations?controller_id=controllerone&vtn_name=vtn1"
{
  "vtnstations": [
    {
      "domain_id": "(DEFAULT)",
      "interface": {},
      "ipaddrs": [
        "10.0.0.2"
      ],
      "macaddr": "b2c3.06b8.2dac",
      "no_vlan_id": "true",
      "port_name": "s2-eth2",
      "station_id": "178195618445172",
      "switch_id": "00:00:00:00:00:00:00:02",
      "vnode_name": "vBridge1",
      "vnode_type": "vbridge",
      "vtn_name": "vtn1"
    },
    {
      "domain_id": "(DEFAULT)",
      "interface": {},
      "ipaddrs": [
```

```

        "10.0.0.1"
    ],
    "macaddr": "ce82.1b08.90cf",
    "no_vlan_id": "true",
    "port_name": "s1-eth1",
    "station_id": "206130278144207",
    "switch_id": "00:00:00:00:00:00:00:01",
    "vnode_name": "vBridge1",
    "vnode_type": "vbridge",
    "vtn_name": "vtn1"
  }
]
}

```

How To View Dataflows in VTN

This example demonstrates on how to view a specific VTN Dataflow information.

Configuration

The same Configuration as Vlan Mapping Example([https://wiki.opendaylight.org/view/OpenDaylight_Virtual_Tenant_Network_\(VTN\):VTN_Coordinator:RestApi:How_to_test_vlan_map_in_Mininet_environment](https://wiki.opendaylight.org/view/OpenDaylight_Virtual_Tenant_Network_(VTN):VTN_Coordinator:RestApi:How_to_test_vlan_map_in_Mininet_environment))

Verification

Get the VTN Dataflows information

```

curl -v -X GET -H 'content-type: application/json' --user 'admin:adminpass'
"http://127.0.0.1:8083/vtn-webapi/dataflows?controller_id=controllerone&
srcmacaddr=924c.e4a3.a743&vlan_id=300&switch_id=00:00:00:00:00:00:00:02&
port_name=s2-eth1"

```

```

{
  "dataflows": [
    {
      "controller_dataflows": [
        {
          "controller_id": "controllerone",
          "controller_type": "odc",
          "egress_domain_id": "(DEFAULT)",
          "egress_port_name": "s3-eth3",
          "egress_station_id": "3",
          "egress_switch_id": "00:00:00:00:00:00:00:03",
          "flow_id": "29",
          "ingress_domain_id": "(DEFAULT)",
          "ingress_port_name": "s2-eth2",
          "ingress_station_id": "2",
          "ingress_switch_id": "00:00:00:00:00:00:00:02",
          "match": {
            "macdstaddr": [
              "4298.0959.0e0b"
            ],
            "macsrcaddr": [
              "924c.e4a3.a743"
            ],
            "vlan_id": [

```

```

        "300"
      ],
      "pathinfos": [
        {
          "in_port_name": "s2-eth2",
          "out_port_name": "s2-eth1",
          "switch_id": "00:00:00:00:00:00:00:02"
        },
        {
          "in_port_name": "s1-eth2",
          "out_port_name": "s1-eth3",
          "switch_id": "00:00:00:00:00:00:00:01"
        },
        {
          "in_port_name": "s3-eth1",
          "out_port_name": "s3-eth3",
          "switch_id": "00:00:00:00:00:00:00:03"
        }
      ]
    },
    "reason": "success"
  }
]
}

```

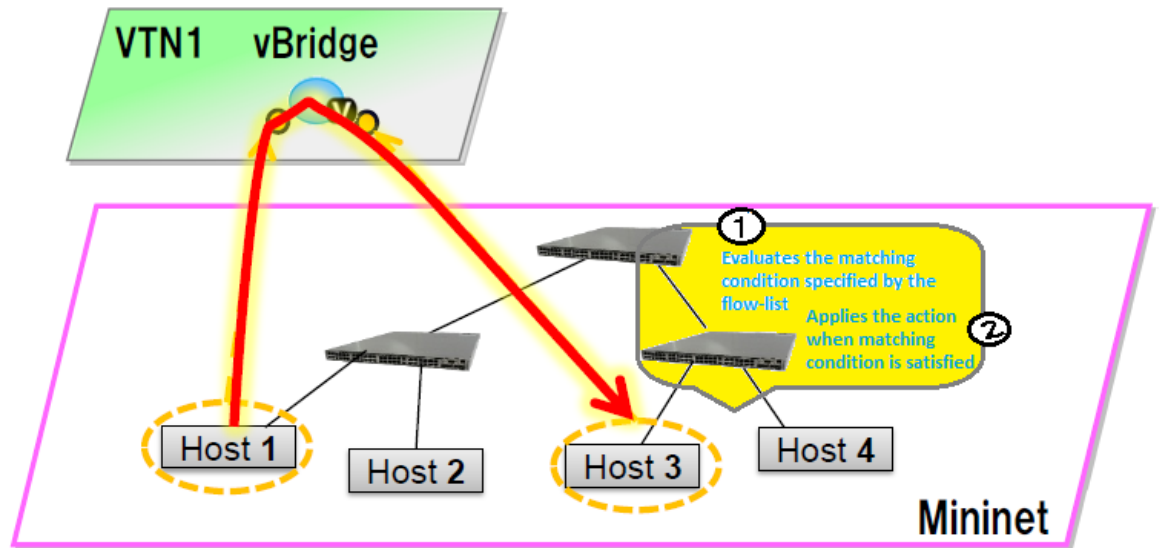
How To Configure Flow Filters Using VTN

Overview

The flow-filter function discards, permits, or redirects packets of the traffic within a VTN, according to specified flow conditions. The table below lists the actions to be applied when a packet matches the condition:

Action	Function
Pass	Permits the packet to pass. As options, packet transfer priority (set priority) and DSCP change (set ip-dscp) is specified.
Drop	Discards the packet.
Redirect	Redirects the packet to a desired virtual interface. As an option, it is possible to change the MAC address when the packet is transferred.

Figure 13.23. Flow Filter



Following steps explain flow-filter function:

- When a packet is transferred to an interface within a virtual network, the flow-filter function evaluates whether the transferred packet matches the condition specified in the flow-list.
- If the packet matches the condition, the flow-filter applies the flow-list matching action specified in the flow-filter.

Requirements

To apply the packet filter, configure the following:

- Create a flow-list and flow-listentry.
- Specify where to apply the flow-filter, for example VTN, vBridge, or interface of vBridge.

Configure mininet and create a topology:

```
$ mininet@mininet-vm:~$ sudo mn --controller=remote,ip=<controller-ip> --topo tree
```

Please generate the following topology

```
$ mininet@mininet-vm:~$ sudo mn --controller=remote,ip=<controller-ip> --topo tree,2
mininet> net
c0
s1 lo: s1-eth1:s2-eth3 s1-eth2:s3-eth3
s2 lo: s2-eth1:h1-eth0 s2-eth2:h2-eth0 s2-eth3:s1-eth1
s3 lo: s3-eth1:h3-eth0 s3-eth2:h4-eth0 s3-eth3:s1-eth2
h1 h1-eth0:s2-eth1
h2 h2-eth0:s2-eth2
h3 h3-eth0:s3-eth1
h4 h4-eth0:s3-eth2
```

Configuration

- Create a controller

```
curl -v --user admin:adminpass -H 'content-type: application/json' -X POST
-d '{"controller": {"controller_id": "controller1", "ipaddr": "10.100.9.61",
" type": "odc", "version": "1.0", "auditstatus": "enable"}}' http://127.0.0.
1:8083/vtn-webapi/controllers
```

- Create a VTN

```
curl -v --user admin:adminpass -H 'content-type: application/json' -X POST -d
'{"vtn" : {"vtn_name": "vtn_one", "description": "test VTN" }}' http://127.0.0.
1:8083/vtn-webapi/vtns.json
```

- Create two vBridges

```
curl -v --user admin:adminpass -H 'content-type: application/json' -X POST
-d '{"vbridge" : {"vbr_name": "vbr_one^C"controller_id": "controller1",
"domain_id": "(DEFAULT)" }}' http://127.0.0.1:8083/vtn-webapi/vtns/vtn_one/
vbridges.json
curl -v --user admin:adminpass -H 'content-type: application/json' -X POST -d
'{"vbridge" :
{"vbr_name": "vbr_two", "controller_id": "controller1",
"domain_id": "(DEFAULT)" }}' http://127.0.0.1:8083/vtn-webapi/vtns/vtn_one/
vbridges.json
```

- Create vBridge interfaces

```
curl -v --user admin:adminpass -H 'content-type: application/json' -X POST -d
'{"interface": {"if_name": "if1", "description": "if_desc1"}}' http://127.0.0.
1:8083/vtn-webapi/vtns/vtn_one/vbridges/vbr_two/interfaces.json
curl -v --user admin:adminpass -H 'content-type: application/json' -X POST -d
'{"interface": {"if_name": "if1", "description": "if_desc1"}}' http://127.0.0.
1:8083/vtn-webapi/vtns/vtn_one/vbridges/vbr_two/interfaces.json
```

- Configure two mappings on the interfaces

```
curl -v --user admin:adminpass -H 'content-type: application/json' -X PUT -d
'{"portmap": {"logical_port_id": "PP-OF:00:00:00:00:00:00:03-s3-eth1"}}'
http://127.0.0.1:8083/vtn-webapi/vtns/vtn_one/vbridges/vbr_two/interfaces/
if1/portmap.json
curl -v --user admin:adminpass -H 'content-type: application/json' -X PUT -d
'{"portmap": {"logical_port_id": "PP-OF:00:00:00:00:00:00:02-s2-eth1"}}'
http://127.0.0.1:8083/vtn-webapi/vtns/vtn_one/vbridges/vbr_two/interfaces/
if2/portmap.json
```

- Create Flowlist

```
curl -v --user admin:adminpass -H 'content-type: application/json' -X POST -d
'{"flowlist": {"fl_name": "flowlist1", "ip_version": "IP"}}' http://127.0.0.
1:8083/vtn-webapi/flowlists.json
```

- Create Flowlistentry

```
curl -v --user admin:adminpass -H 'content-type: application/json' -X
POST -d '{"flowlistentry": {"seqnum": "233", "macethertype": "0x8000",
"ipdstaddr": "10.0.0.3", "ipdstaddrprefix": "2", "ipsrcaddr": "10.0.0.2",
"ipsrcaddrprefix": "2", "ipproto": "17", "ipdscp": "55", "icmptypenum": "232",
```

```
"icmpcodenum": "232"}}' http://127.0.0.1:8083/vtn-webapi/flowlists/flowlist1/flowlistentries.json
```

- Create vBridge Interface Flowfilter

```
curl -v --user admin:adminpass -X POST -H 'content-type: application/json' -d '{"flowfilter": {"ff_type": "in"}}' http://127.0.0.1:8083/vtns/vtn_one/vbridges/vbr_two/interfaces/if1/flowfilters.json
```

Flow filter demonstration with DROP action-type

```
curl -v --user admin:adminpass -X POST -H 'content-type: application/json' -d '{"flowfilterentry": {"seqnum": "233", "fl_name": "flowlist1", "action_type": "drop", "priority": "3", "dscp": "55" }}' http://127.0.0.1:8083/vtn-webapi/vtns/vtn_one/vbridges/vbr_two/interfaces/if1/flowfilters/in/flowfilterentries.json
```

Verification

As we have applied the action type "drop", ping should fail.

```
mininet> h1 ping h3
PING 10.0.0.2 (10.0.0.2) 56(84) bytes of data.
From 10.0.0.1 icmp_seq=1 Destination Host Unreachable
From 10.0.0.1 icmp_seq=2 Destination Host Unreachable
```

In controller you can see the DROP action type information as below, here action as DROP.

```
osgi> readflows 0000000000000003
```

```
[FlowOnNode[flow =Flow[match = Match [fields={DL_VLAN=DL_VLAN(0), IN_PORT=IN_PORT(OF|1@OF|00:00:00:00:00:00:00:03), DL_DST=DL_DST(4e:08:1d:a6:05:08), DL_SRC=DL_SRC(be:15:00:a4:96:13)}, matches=15], actions = [DROP], priority = 10, id = 0, idleTimeout = 0, hardTimeout = 300], tableId = 0, sec = 18, nsec = 475000000, pkt = 20, byte = 1232], FlowOnNode[flow =Flow[match = Match [fields={DL_VLAN=DL_VLAN(0), IN_PORT=IN_PORT(OF|3@OF|00:00:00:00:00:00:00:03), DL_DST=DL_DST(be:15:00:a4:96:13), DL_SRC=DL_SRC(4e:08:1d:a6:05:08)}, matches=15], actions = [OUTPUT[OF|1@OF|00:00:00:00:00:00:00:03]], priority = 10, id = 0, idleTimeout = 0, hardTimeout = 0], tableId = 0, sec = 18, nsec = 489000000, pkt = 10, byte = 812]]
```

Flow filter demonstration with PASS action-type

```
curl -v --user admin:adminpass -X PUT -H 'content-type: application/json' -d '{"flowfilterentry": {"seqnum": "233", "fl_name": "flowlist1", "action_type": "pass", "priority": "3", "dscp": "55" }}' http://127.0.0.1:8083/vtn-webapi/vtns/vtn_one/vbridges/vbr_two/interfaces/if1/flowfilters/in/flowfilterentries/233.json
```

Verification

```
mininet> h1 ping h3
PING 10.0.0.3 (10.0.0.3) 56(84) bytes of data.
64 bytes from 10.0.0.3: icmp_req=1 ttl=64 time=0.984 ms
64 bytes from 10.0.0.3: icmp_req=2 ttl=64 time=0.110 ms
64 bytes from 10.0.0.3: icmp_req=3 ttl=64 time=0.098 ms
```

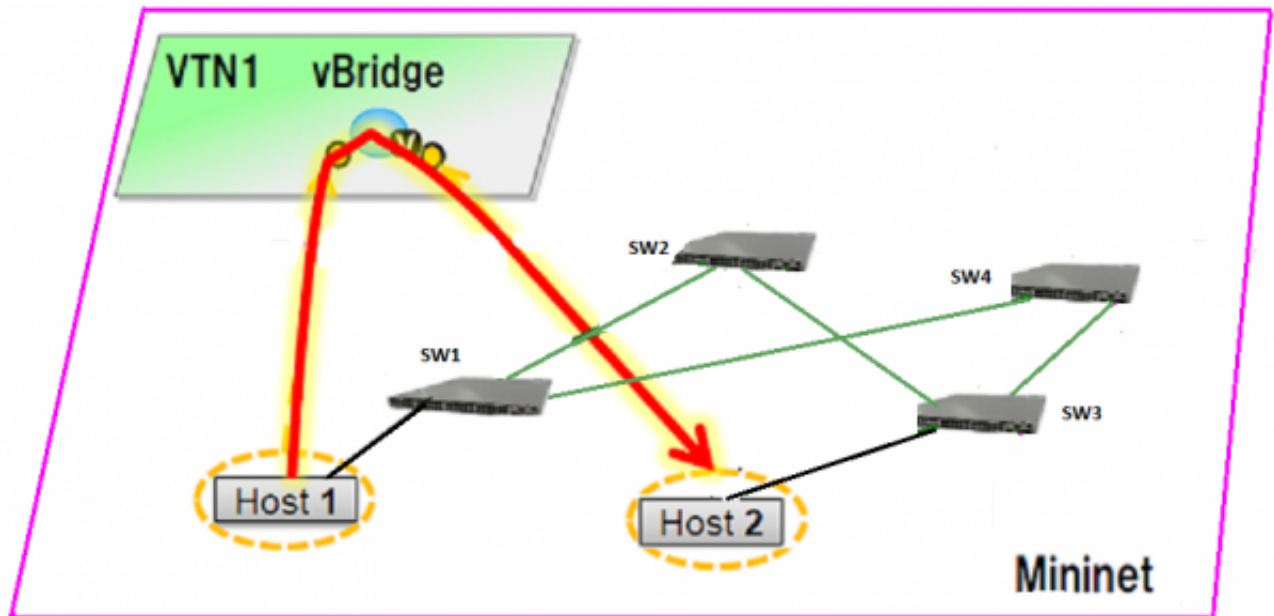
In controller you can see the PASS action type information by executing the following command:

```
osgi> readflows 00000000000000003
```

How To Use VTN To Make Packets Take Different Paths

This example demonstrates on how to create a specific VTN Path Map information.

Figure 13.24. PathMap



Requirement

- Save the mininet script given below as pathmap_test.py and run the mininet script in the mininet environment where Mininet is installed.
- Create topology using the below mininet script:

```
from mininet.topo import Topo
class MyTopo( Topo ):
    "Simple topology example."
    def __init__( self ):
        "Create custom topo."
        # Initialize topology
        Topo.__init__( self )
        # Add hosts and switches
        leftHost = self.addHost( 'h1' )
        rightHost = self.addHost( 'h2' )
        leftSwitch = self.addSwitch( 's1' )
        middleSwitch = self.addSwitch( 's2' )
        middleSwitch2 = self.addSwitch( 's4' )
        rightSwitch = self.addSwitch( 's3' )
        # Add links
        self.addLink( leftHost, leftSwitch )
        self.addLink( leftSwitch, middleSwitch )
        self.addLink( leftSwitch, middleSwitch2 )
        self.addLink( middleSwitch, rightSwitch )
```



```

        self.addLink( middleSwitch2, rightSwitch )
        self.addLink( rightSwitch, rightHost )
    topos = { 'mytopo': ( lambda: MyTopo() ) }

```

```

mininet> net
c0
s1 lo:  s1-eth1:h1-eth0 s1-eth2:s2-eth1 s1-eth3:s4-eth1
s2 lo:  s2-eth1:s1-eth2 s2-eth2:s3-eth1
s3 lo:  s3-eth1:s2-eth2 s3-eth2:s4-eth2 s3-eth3:h2-eth0
s4 lo:  s4-eth1:s1-eth3 s4-eth2:s3-eth2
h1 h1-eth0:s1-eth1
h2 h2-eth0:s3-eth3

```

- Generate traffic by pinging between hosts h1 and h2 before creating the portmaps respectively

```

mininet> h1 ping h2
PING 10.0.0.2 (10.0.0.2) 56(84) bytes of data.
From 10.0.0.1 icmp_seq=1 Destination Host Unreachable
From 10.0.0.1 icmp_seq=2 Destination Host Unreachable
From 10.0.0.1 icmp_seq=3 Destination Host Unreachable
From 10.0.0.1 icmp_seq=4 Destination Host Unreachable

```

Configuration

- Create Controller

```

curl --user admin:adminpass -H 'content-type: application/json' -X POST -d '{"controller": {"controller_id": "odc", "ipaddr": "10.100.9.42", "type": "odc", "version": "1.0", "auditstatus": "enable"}}' http://127.0.0.1:8083/vtn-webapi/controllers.json

```

- Create a VTN

```

curl --user admin:adminpass -H 'content-type: application/json' -X POST -d '{"vtn" : {"vtn_name": "vtn1", "description": "test VTN" }}' http://127.0.0.1:8083/vtn-webapi/vtns.json

```

- Create a vBridge in the VTN

```

curl --user admin:adminpass -H 'content-type: application/json' -X POST -d '{"vbridge" : {"vbr_name": "vBridge1", "controller_id": "odc", "domain_id": "(DEFAULT)" }}' http://127.0.0.1:8083/vtn-webapi/vtns/vtn1/vbridges.json

```

- Create two Interfaces into the vBridge

```

curl --user admin:adminpass -H 'content-type: application/json' -X POST -d '{"interface": {"if_name": "if1", "description": "if_desc1"}}' http://127.0.0.1:8083/vtn-webapi/vtns/vtn1/vbridges/vBridge1/interfaces.json
curl --user admin:adminpass -H 'content-type: application/json' -X POST -d '{"interface": {"if_name": "if2", "description": "if_desc2"}}' http://127.0.0.1:8083/vtn-webapi/vtns/vtn1/vbridges/vBridge1/interfaces.json

```

- Configure two mappings on the interfaces

```

curl --user admin:adminpass -H 'content-type: application/json' -X PUT -d '{"portmap": {"logical_port_id": "PP-OF:00:00:00:00:00:00:00:01-s1-eth1"}}'

```

```

http://127.0.0.1:8083/vtn-webapi/vtns/vtn1/vbridges/vBridge1/interfaces/if1/
portmap.json
curl --user admin:adminpass -H 'content-type: application/json' -X PUT -d
'{"portmap":{"logical_port_id": "PP-OF:00:00:00:00:00:00:03-s3-eth3"}}'
http://127.0.0.1:8083/vtn-webapi/vtns/vtn1/vbridges/vBridge1/interfaces/if2/
portmap.json

```

- Generate traffic by pinging between hosts h1 and h2 after creating the portmaps respectively

```

mininet> h1 ping h2
PING 10.0.0.2 (10.0.0.2) 56(84) bytes of data.
64 bytes from 10.0.0.2: icmp_req=1 ttl=64 time=36.4 ms
64 bytes from 10.0.0.2: icmp_req=2 ttl=64 time=0.880 ms
64 bytes from 10.0.0.2: icmp_req=3 ttl=64 time=0.073 ms
64 bytes from 10.0.0.2: icmp_req=4 ttl=64 time=0.081 ms

```

- Get the VTN Dataflows information

```

curl -X GET -H 'content-type: application/json' --user 'admin:adminpass'
"http://127.0.0.1:8083/vtn-webapi/dataflows?&switch_id=
00:00:00:00:00:00:00:01&port_name=s1-eth1&controller_id=odc&srcmacaddr=de3d.
7dec.e4d2&no_vlan_id=true"

```

- Create a Flowcondition in the VTN

```

curl --user admin:admin -H 'content-type: application/json' -X PUT
-d '{"name": "flowcond_1","match": [{"index": 1,"ethernet": {"src":
"ca:9e:58:0c:1e:f0","dst": "ba:bd:0f:e3:a8:c8","type": 2048},"inetMatch":
{"inet4": {"src": "10.0.0.1","dst": "10.0.0.2","protocol": 1}}]}' http://10.
100.9.42:8080/controller/nb/v2/vtn/default/flowconditions/flowcond_1

```

- Create a Pathmap in the VTN

```

curl --user admin:admin -H 'content-type: application/json' -X PUT -d
'{"index": 10, "condition":"flowcond_1", "policy":1, "idleTimeout": 300,
"hardTimeout": 0}' http://10.100.9.42:8080/controller/nb/v2/vtn/default/
pathmaps/1

```

- Get the Path policy information

```

curl --user admin:admin -H 'content-type: application/json' -X GET -d
'{"id": 1,"default": 100000,"cost": [{"location": {"node": {"type":
"OF","id": "00:00:00:00:00:00:00:01"},"port": {"type": "OF","id": "3",
"name": "s1-eth3"}}, {"location": {"node": {"type": "OF",
"id": "00:00:00:00:00:00:00:04"},"port": {"type": "OF","id": "2","name":
"s4-eth2"}}, {"location": {"node": {"type": "OF", "id":
"00:00:00:00:00:00:00:03"},"port": {"type": "OF","id": "3","name": "s3-
eth3"}}, {"cost": 100000}]}]' http://10.100.9.42:8080/controller/nb/v2/vtn/
default/pathpolicies/1

```

Verification

- Before applying Path policy information in the VTN

```

{
  "pathinfos": [
    {
      "in_port_name": "s1-eth1",

```

```

        "out_port_name": "s1-eth2",
        "switch_id": "00:00:00:00:00:00:00:01"
    },
    {
        "in_port_name": "s2-eth1",
        "out_port_name": "s2-eth2",
        "switch_id": "00:00:00:00:00:00:00:02"
    },
    {
        "in_port_name": "s3-eth1",
        "out_port_name": "s3-eth3",
        "switch_id": "00:00:00:00:00:00:00:03"
    }
    ]
}

```

- After applying Path policy information in the VTN

```

{
  "pathinfos": [
    {
      "in_port_name": "s1-eth1",
      "out_port_name": "s1-eth3",
      "switch_id": "00:00:00:00:00:00:00:01"
    },
    {
      "in_port_name": "s4-eth1",
      "out_port_name": "s4-eth2",
      "switch_id": "00:00:00:00:00:00:00:04"
    },
    {
      "in_port_name": "s3-eth2",
      "out_port_name": "s3-eth3",
      "switch_id": "00:00:00:00:00:00:00:03"
    }
  ]
}

```

VTN Coordinator(Troubleshooting HowTo)

Overview

This page demonstrates Installation troubleshooting steps of VTN coordinator. OpenDaylight VTN provides multi-tenant virtual network functions on OpenDaylight controllers. OpenDaylight VTN consists of two parts:

- VTN Coordinator
- VTN Manager.

VTN Coordinator orchestrates multiple VTN Managers running in OpenDaylight Controllers, and provides VTN Applications with VTN API. VTN Manager is OSGi bundles running in OpenDaylight Controller. Current VTN Manager supports only OpenFlow switches. It handles PACKET_IN messages, sends PACKET_OUT messages, manages host information, and installs flow entries into OpenFlow switches to provide VTN Coordinator with virtual network functions. The requirements for installing these two are different. Therefore, we recommend that you install VTN Manager and VTN Coordinator in different machines.

List of installation Troubleshooting How to's

How to install VTN Coordinator?

- [https://wiki.opendaylight.org/view/OpenDaylight_Virtual_Tenant_Network_\(VTN\):Installation:VTN_Coordinator](https://wiki.opendaylight.org/view/OpenDaylight_Virtual_Tenant_Network_(VTN):Installation:VTN_Coordinator)

After executing db_setup, you have encountered the error "Failed to setup database"?

The error could be due to the below reasons * Access Restriction

The user who owns /usr/local/vtn/ directory and installs VTN coordinator, can only start db_setup. Example :

```
The directory should appear as below (assuming the user as "vtn"):  
# ls -l /usr/local/  
  drwxr-xr-x. 12 vtn  vtn  4096 Mar 14 21:53 vtn  
If the user doesnot own /usr/local/vtn/ then, please run the below command  
(assuming the username as vtn),  
      chown -R vtn:vtn /usr/local/vtn
```

- Postgres not Present

```
1. In case of Fedora/CentOS/RHEL, please check if /usr/pgsql/<version>  
directory is present and also ensure the commands initdb, createdb,pg_ctl,  
psql are working. If, not please re-install postgres packages  
2. In case of Ubuntu, check if /usr/lib/postgres/<version> directory is  
present and check for the commands as in the previous step.
```

- Not enough space to create tables

Please check df -k and ensure enough free space is available.

- If the above steps do not solve the problem, please refer to the log file for the exact problem

/usr/local/vtn/var/dbm/unc_setup_db.log for the exact error.

What are the things to check after vtn_start?

- list of coordinator processes
- Run the below command ensure the Coordinator daemons are running.

```
Command: /usr/local/vtn/bin/unc_dmctl status  
Name      Type      IPC Channel  PID  
-----  
  drvodcd  DRIVER    drvodcd     15972  
  lgenwd   LOGICAL   lgenwd      16010  
  phynwd   PHYSICAL  phynwd      15996
```

- Issue the curl command to fetch version and ensure the process is able to respond.

How to debug a startup failure? The following activities take place in order during startup * Database server is started after setting virtual memory to required value,Any database startup errors will be reflected in any of the below logs.

```

/usr/local/vtn/var/dbm/unc_db_script.log.
/usr/local/vtn/var/db/pg_log/postgresql-*.log (the pattern will have
the date)

```

- uncd daemon is kicked off, The daemon in turn kicks off the rest of the daemons.

```

Any uncd startup failures will be reflected in /usr/local/vtn/var/uncd/
uncd_start.err.

```

After setting up the apache tomcat server, what are the aspects that should be checked.

Please check if catalina is running.

```

The command ps -ef | grep catalina | grep -v grep should list a catalina
process

```

If you encounter an erroneous situation where the REST API is always failing.

```

Please ensure the firewall settings for port:8080(Helium release) or
port:8083(Post Helium release) and enable the same.

```

How to debug a REST API returning a failure message? Please check the `/usr/share/java/apache-tomcat-7.0.39/logs/core/core.log` for failure details.

REST API for VTN configuration fails, how to debug? The default log level for all daemons is "INFO", to debug the situation TRACE or DEBUG logs may be needed. To increase the log level for individual daemons, please use the commands suggested below

```

/usr/local/vtn/bin/lgcnw_control loglevel trace -- upll daemon log
/usr/local/vtn/bin/phynw_control loglevel trace -- uppl daemon log
/usr/local/vtn/bin/unc_control loglevel trace -- uncd daemon log
/usr/local/vtn/bin/drvodc_control loglevel trace -- Driver daemon log

```

After setting the log levels, the operation can be repeated and the log files can be referred for debugging.

Problems while Installing PostgreSQL due to openssl. Errors may occur when trying to install postgresql rpms. Recently PostgreSQL has upgraded all their binaries to use the latest openssl versions with fix for <http://en.wikipedia.org/wiki/Heartbleed> Please upgrade the openssl package to the latest version and re-install. For RHEL 6.1/6.4 : If you have subscription, Please use the same and update the rpms. The details are available in the following link * <https://access.redhat.com/site/solutions/781793> ACCESS-REDHAT

```

rpm -Uvh http://mirrors.kernel.org/centos/6/os/x86_64/Packages/openssl-1.0.
1e-15.el6.x86_64.rpm
rpm -ivh http://mirrors.kernel.org/centos/6/os/x86_64/Packages/openssl-
devel-1.0.1e-15.el6.x86_64.rpm

```

For other linux platforms, Please do yum update, the public respositroes will have the latest openssl, please install the same.