

EFFICIENT TRAFFIC MANAGEMENT IN SOFTWARE-DEFINED NETWORKING

TUSHAR AHERKAR	<h20171030084@hyderabad.bitspilani.ac.in>
DAMINI NANAWARE	<h20171030088@hyderabad.bits-pilani.ac.in>
SOURABH SETHI	<h20171030072@hyderabad.bits-pilani.ac.in>

1. Introduction:

SDN is intended to address the way that the static architecture of conventional networks doesn't support the dynamic, scalable computing and storage needs of more present day computing environments such as data centres. This is done by decoupling or disassociating the system that makes decisions about where traffic is sent (the control plane) from the underlying systems that forward traffic to the intended destination (the data plane). The Software Defined Networking method centralizes control of the network by separating the control logic to off-device computer resources.

An SDN controller is an application in software-defined networking that manages flow control to enable intelligent networking. SDN controllers are based on protocols, such as OpenDayLight, that allows servers to tell switches where to send packets.

OpenDaylight: OpenDaylight is an exceedingly accessible, measured, extensible, versatile and multi-convention controller framework worked for SDN arrangements on current heterogeneous multivendor systems.

OpenDaylight gives a model-driven service abstraction platform that enables clients to compose applications that effectively work over a wide range of hardware and southbound APIs. These applications utilize the controller to accumulate network intelligence, run algorithms to perform some analysis, and afterward utilize the controller to arrange the new rules, if any, throughout the network.

Mininet Emulator: Mininet is a product emulator for prototyping a large network on to a single machine. Mininet emulator is a cheap and rapidly configurable system testbed. It enables the client to rapidly make, cooperate with, customize and share a software-defined network (SDN) prototype to simulate a network.

Mininet utilizes virtual has, changes, and connections to make a system on a single OS part, and uses the real network stack to process packets and connect to real networks. Moreover, Unix/Linux-based applications are likewise permitted to keep running on virtual hosts. In an OpenDayLight arrangement simulated by Mininet, a real OpenDayLight controller application can keep running on another machine or on a similar machine where virtual hosts are emulated.

OVSDB: OVS Databse is on mininet VM and its common to all the switches. We can connect to this OVSDB in two ways:

- a. Active mode:** ODL initially connects to OVSDB and after that it will listen to changes on OVSDB.
- b. Passive mode:** OVSDB will listen for changes from ODL. We are making use of this mode.

In OVSDb we use following tables

Port Table:

This table has details of all the interface which connect to corresponding hosts and each interface is associated with Qos foreign key(UUID of QoS table). Each tuple is uniquely identified by UUID.

Qos Table:

This table contains Quality of Service (QoS) configuration for each Port that references it. Each row is identified by UUID and is mapped to several Queue in Queue table.

Queue Table:

This table contains a configuration for a port output queue, used in configuring Quality of Service (QoS) features. We can set max_rate and min_rate using Other-config column. We used max_rate to set the maximum bandwidth which user request and min_rate to some predefined value.

Restconf Api:

We Used this Api to set flows.

While setting the flow we define set queue in the action to the queue number in QoS Table. set_queue is always zero because we have only one queue associated with each port.

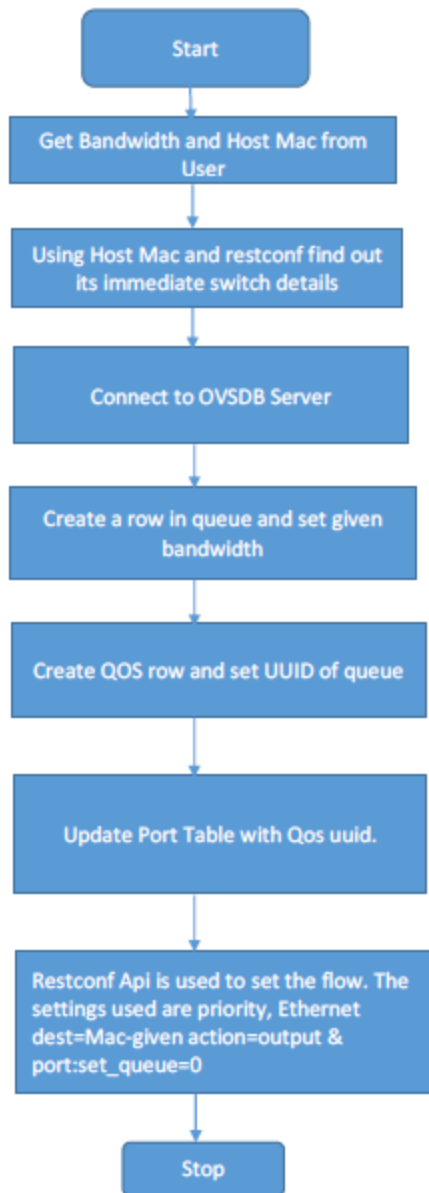
Northbound Api:

Northbound Apis are used to access OVSDb on mininet.

2. Project Objective:

The primary aim of the Project is to understand Software Defined Networks and usage of Mininet and OpenDayLight controller. Along with this we aim to shape traffic according to requirement using two methods which are manual control and Restconf API. Simulation is performed in using Mininet and traffic management based on the topology is done using OpenDayLight controller.

3. Work Flow:



4. Tools used and Installation overview:

Two tools are mainly used till now in the project to create the SDN environment. These are :

- a . Mininet
- b. OpenDayLight

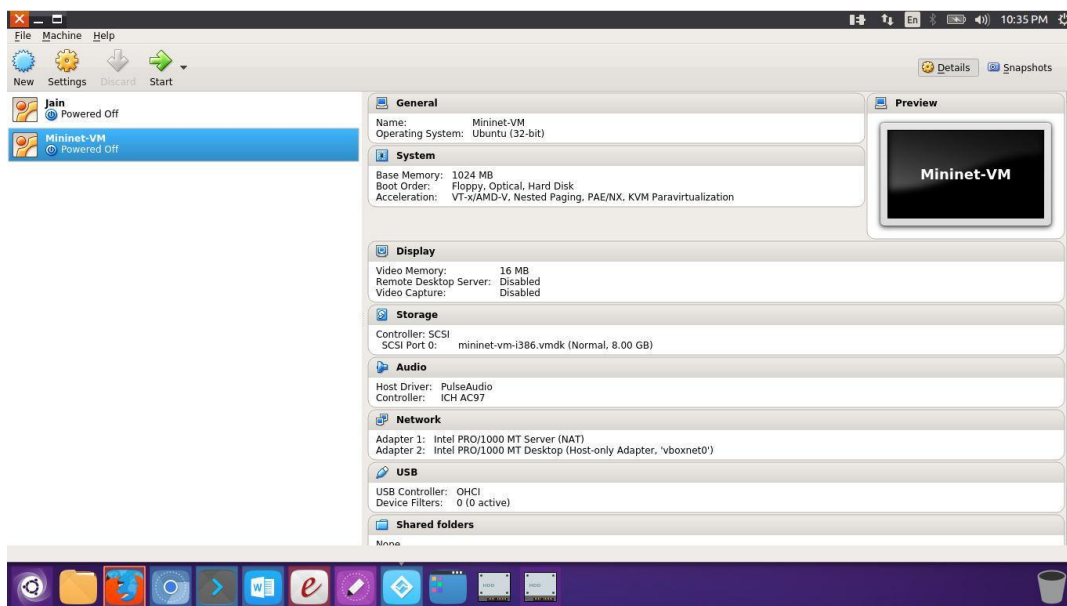
Mininet - Mininet is a network simulator used to create scalable SDNs using Linux processes. Mininet is used in this project to simulate the topology and test the traffic flows.

OpenDayLight – Topology created using mininet emulator is viewed using OpenDayLight controller. All the details regarding host , switches ,and their variation is customized using this software protocol

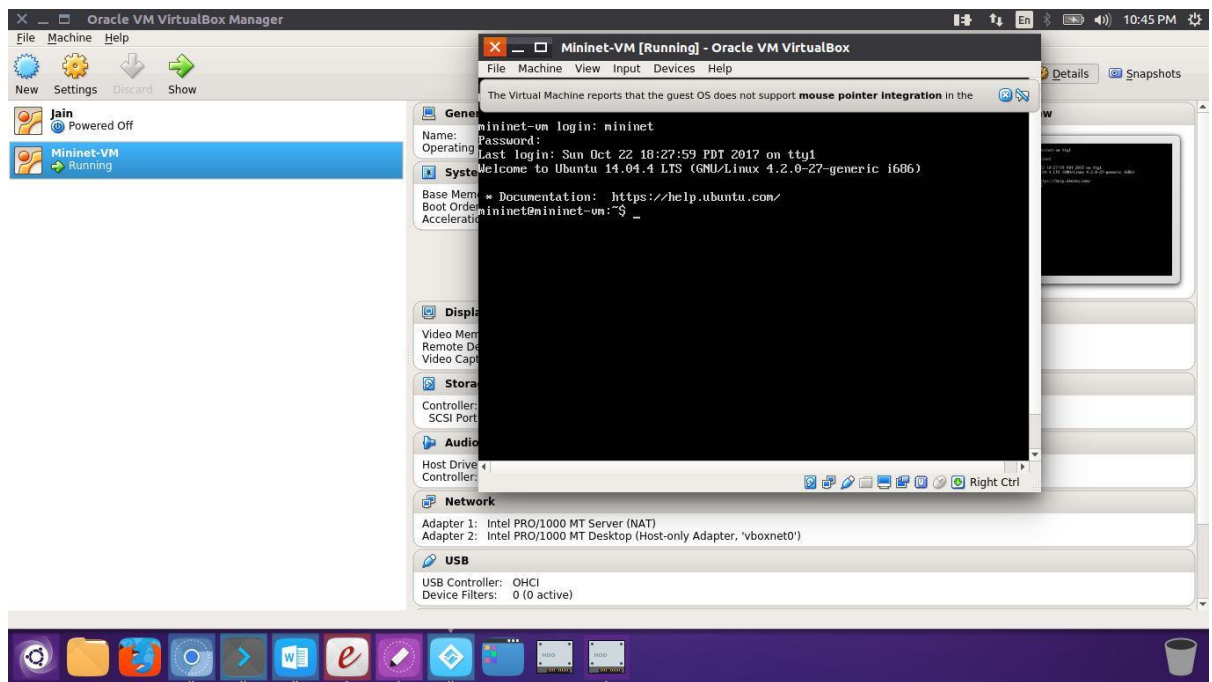
Installation of mininet guide:

We have installed a mininet virtualbox over oracle virtual box . Reference Link is shared at the end [1] where detailed commands being used are mentioned. Following are the steps followed to install mininet virtualbox :

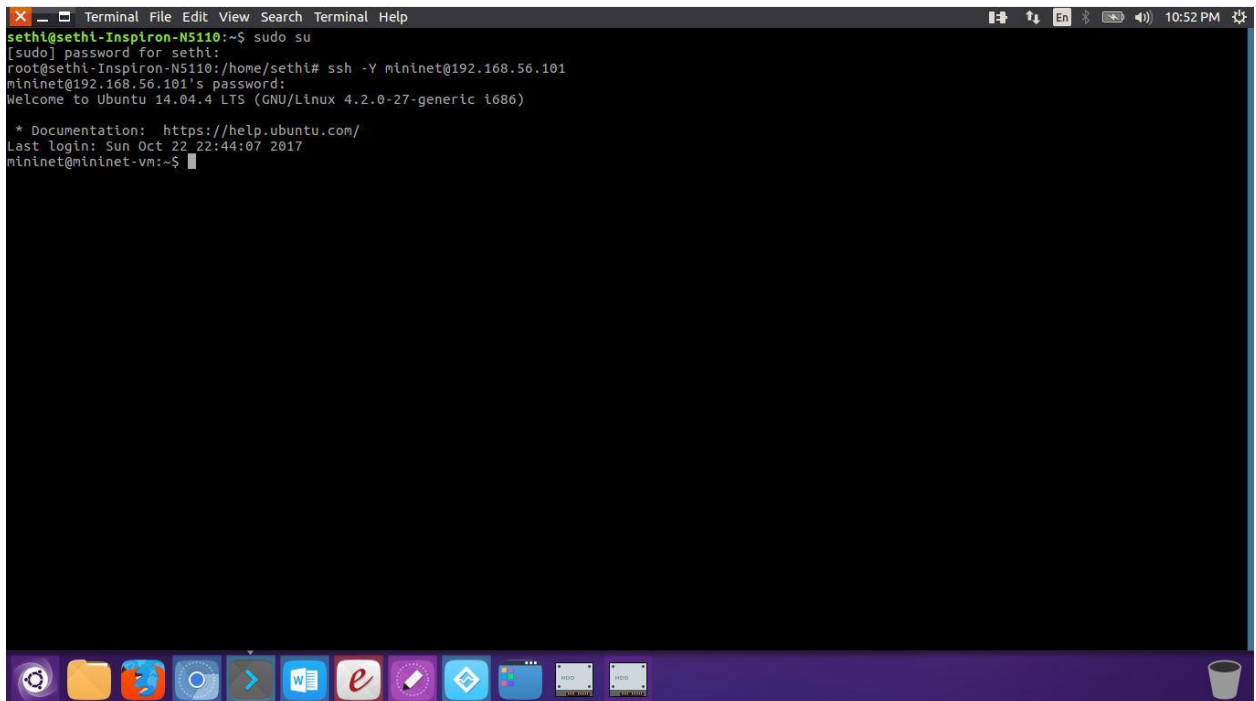
1. Downloaded the Mininet virtual machine.
2. Import the virtual machine into VirtualBox
3. Add a Host-only Adapter in VirtualBox
4. Add Network Adapter to Mininet virtual machine
5. Start the Mininet virtual machine



6. Once started , Login ID and password are required which are “mininet” by default as shown below are used :



7.Using SSH to connect to the Mininet VM as shown below from host machine :



Here mininet has been logged in using our host machine using command
ssh -Y mininet@192.168.56.101

192.168.56.101 is the mininet Virtual machine IP .

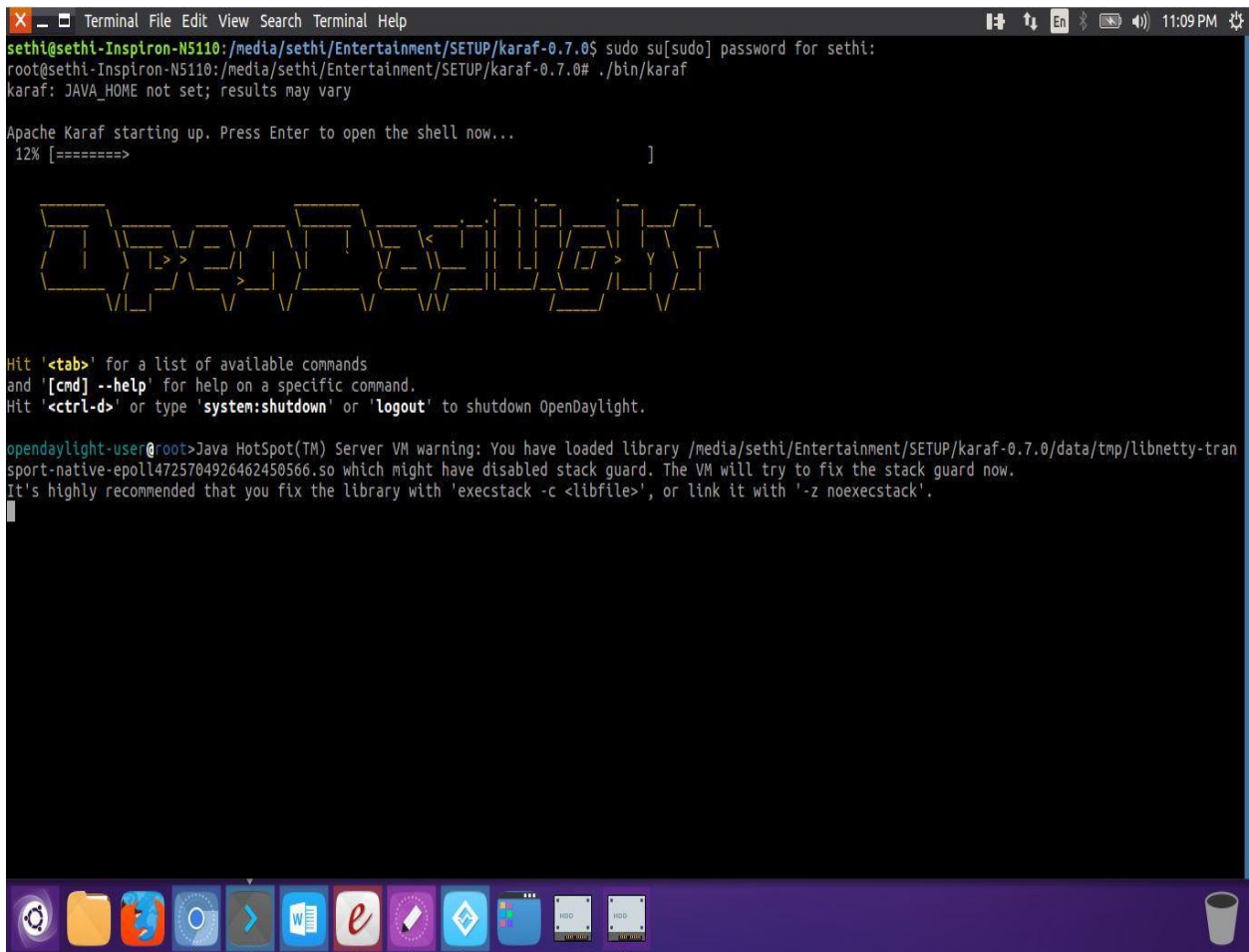
Hence , by now , our emulator has been installed successfully .

Installation of OpenDayLight guide :

We have installed a OpenDayLight controller for observing the network topology created. Reference Link is shared at the end [2] in reference section where detailed commands being used are mentioned.

Following are the steps followed to install OpenDayLight controller :

1. Downloaded the OpenDayLight latest version (Nitrogen)from <https://www.opendaylight.org/>
2. Installed OpenDaylight using the steps shared in reference link[2]
3. Start OpenDaylightas shown below using command :
./bin/karaf (In the karaf directory path where files are downloaded)



```
sethi@sethi-Inspiron-N5110: /media/sethi/Entertainment/SETUP/karaf-0.7.0$ sudo su[sudo] password for sethi:
root@sethi-Inspiron-N5110: /media/sethi/Entertainment/SETUP/karaf-0.7.0# ./bin/karaf
karaf: JAVA_HOME not set; results may vary

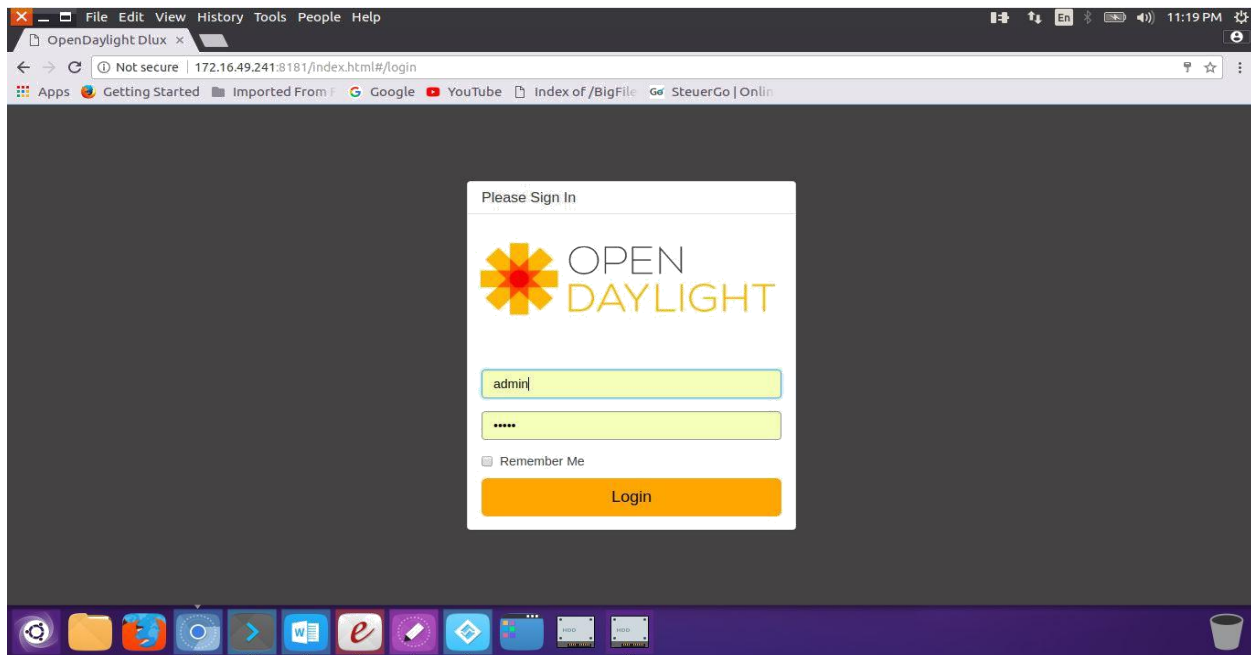
Apache Karaf starting up. Press Enter to open the shell now...
12% [=====>]

Hit '<tab>' for a list of available commands
and '[cmd] --help' for help on a specific command.
Hit '<ctrl-d>' or type 'system:shutdown' or 'logout' to shutdown OpenDaylight.

opendaylight-user@root>Java HotSpot(TM) Server VM warning: You have loaded library /media/sethi/Entertainment/SETUP/karaf-0.7.0/data/tmp/libnetty-tran
sport-native-epoll4725704926462450566.so which might have disabled stack guard. The VM will try to fix the stack guard now.
It's highly recommended that you fix the library with 'execstack -c <libfile>', or link it with '-z noexecstack'.
```

4. Now run the webapplication of OpenDayLight from our browser using below mentio0ned link
<http://172.16.49.241:8181/index.html#/node/index>

Here , 172.16.49.241 is IP of machine where controller has been installed



Hence the environmental requirements for creating the topology has been successfully done.

5 Methodology

Now that we have finished installation let, we can start with first step of implementation:

5.1 Building a topology on Mininet

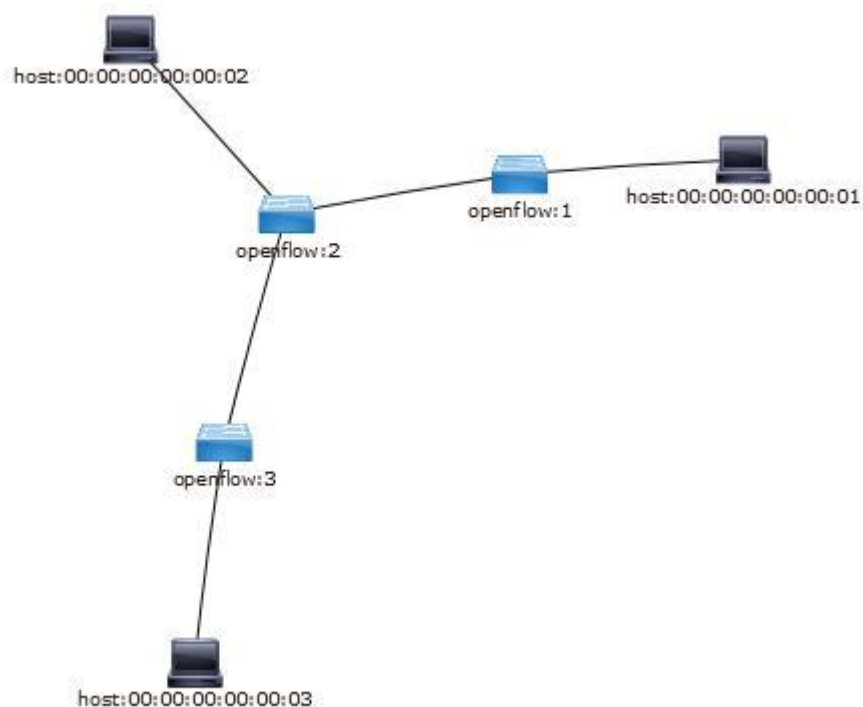
1. Let us consider a simple linear topology of three switches
2. Each switch is connected to one host
3. The MAC address on each host can be set as a simple number
4. The remote controller, OpenDaylight, is at IP address 192.168.217.135:6633

Following image shows the command used in order to create the linear topology.

The default topology is the **minimal** topology, which includes one OpenFlow kernel switch connected to two hosts, plus the OpenFlow reference controller. This topology could also be specified on the command line with **-topo minimal**. Here we chose **linear** topology.

```
tushar@ubuntu:~$ sudo mn --topo linear,3 --mac --controller=remote,ip=192.168.217.135,port=6633 --switch ovsk,protocols=OpenFlow13
*** Creating network
*** Adding controller
*** Adding hosts:
h1 h2 h3
*** Adding switches:
s1 s2 s3
*** Adding links:
(h1, s1) (h2, s2) (h3, s3) (s1, s2) (s2, s3)
*** Configuring hosts
h1 h2 h3
*** Starting controller
*** Starting 3 switches
s1 s2 s3
*** Starting CLI:
```

We can now see this topology in our OpenDaylight interface. We can see three nodes and Openflow switches connected to a OpenDaylight controller.



5.2 Testing the created network:

Now to test our connections we ping all the created nodes. Every host should be able to reach every other host.

```
mininet> pingall
*** Ping: testing ping reachability
h1 -> h2 h3
h2 -> h1 h3
h3 -> h1 h2
*** Results: 0% dropped (6/6 received)
```

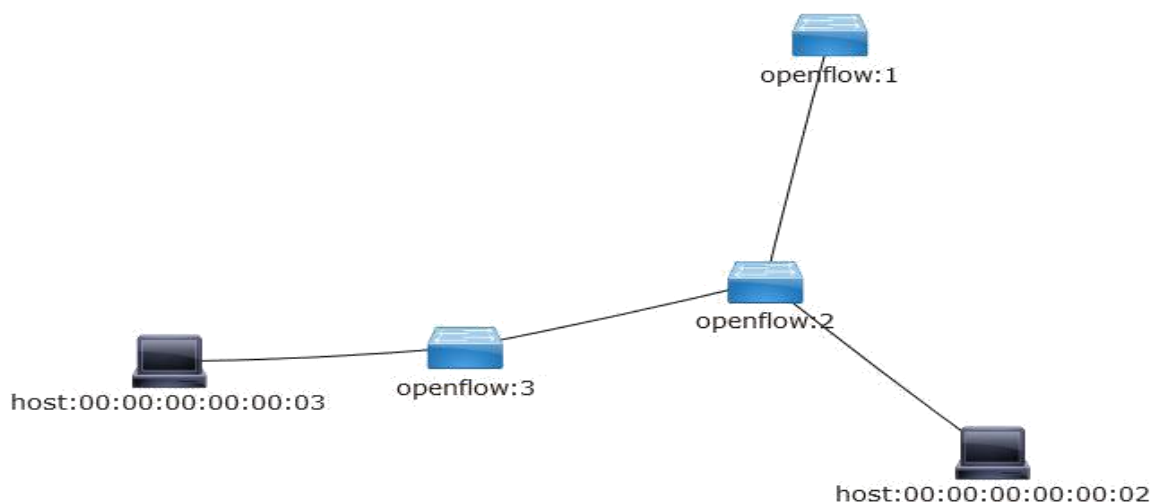
Next we can try different commands to check number of nodes

```
mininet> nodes
available nodes are:
c0 h1 h2 h3 s1 s2 s3
```

Now we experimented to turn down a link and then tried pinging the nodes. We turn down link between h1 and s1 using following command and then ping all.

```
mininet> link h1 s1 down
mininet> pingall
*** Ping: testing ping reachability
h1 -> X X
h2 -> X h3
h3 -> X h2
*** Results: 66% dropped (2/6 received)
```

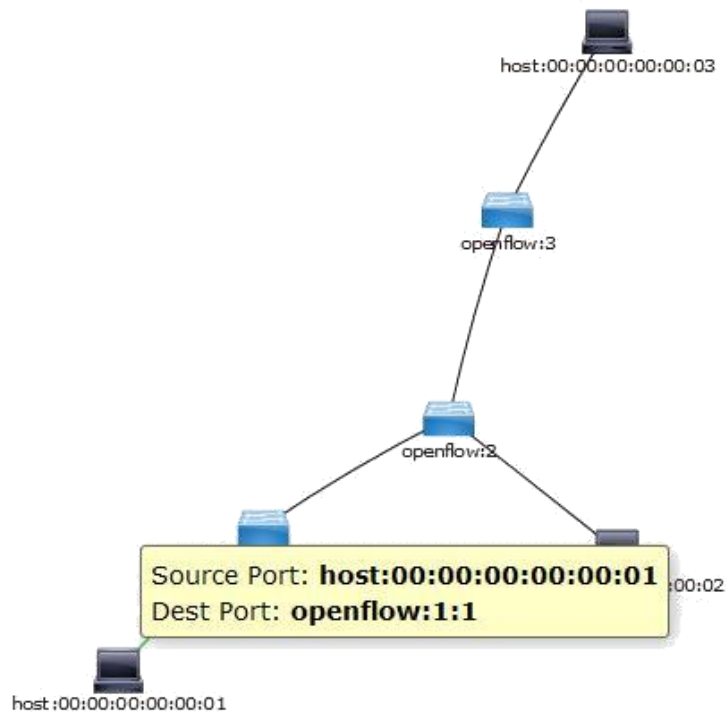
As we can see we now cannot reach all the from h1 and cannot reach h1 through h2 and h3. Below figure shows view in the OpenDaylight interface where we can see h1 is not pre



We can again connect this link and then check pinging the nodes.

```
mininet> link h1 s1 up
mininet> pingall
*** Ping: testing ping reachability
h1 -> h2 h3
h2 -> h1 h3
h3 -> h1 h2
*** Results: 0% dropped (6/6 received)
```

Following are the results shown in the OpenDaylight interface



The above figure shows host h1 and link between h1 and s1 again being connected.

5.3 Traffic Shaping

We have used two approaches on our project:

APPROACH 1: Manually shaping the traffic using Open_vSwitch:

Commands for achieving work flow:

```
mininet@mininet-vm:~$ sudo ovs-vsctl set port s1-eth1 qos=@newqos -- --id=@newqos create qos type=linux-htb other-config:max-rate=5000000 queue
s=1=@q1 -- --id=@q1 create queue other-config:min-rate=500000 other-config:max-rate=5000000
37115d01-61aa-4609-a2bf-7834d3dcec8b
b5f36cc5-fcf9-4272-bfcb-0b3f1965b6d3
```

Results after executing above command:

\$sudo ovsdb-client dump Open_vSwitch|more

```
mininet@mininet-vm: ~
_uuid      lacp mac name      other_config qos      bond_downdelay bond_fake_iface bond_mode bond_updelay external_ids fake_bridge interfaces
-----
b85c62de-4ddf-4178-a79f-5736f71f0e72 0      false      []      0      {}      {}      {}      false      [fde234c2-4c94-45a0-a0cb-de
208fa1e1ea] []      "s1"      {}      []      false      []      0      {}      {}      {}      false      [c0df5f92-d106-402d-b9ef-77
9621f12b-8abf-49db-9ec9-b043bae8716e 0      false      []      0      {}      {}      {}      false      [80116bf5-c1e1-4e0f-8a31-79
f51fd9309d] []      "s1-eth1" {}      37115d01-61aa-4609-a2bf-7834d3dcec8b {}      {}      {}      false      [38e147db-a383-44e0-9314-2a
ebc70419-e1c8-4306-8495-57799f73cd3d 0      false      []      0      {}      {}      {}      false      [9bcabc3-1fe2-46c0-8a38-13
e94be7d05f] []      "s1-eth2" {}      []      false      []      0      {}      {}      {}      false      [6f9cc16e-39a2-4707-bee4-49
8c1af60e-1d5b-4f21-9380-694abccaca36 0      false      []      0      {}      {}      {}      false      [52cc0226-31b3-4874-8930-c2
e40da2ec04] []      "s2"      {}      []      false      []      0      {}      {}      {}      false      [7472bdbc-1230-4adf-b863-47
b8ce5f23-f221-486f-aa75-8d8a11475ee0 0      false      []      0      {}      {}      {}      false      [25aa243b-006d-4475-8a50-eb
91cf20c96c] []      "s2-eth1" {}      []      false      []      0      {}      {}      {}      false      [208a1ff4-71f1-48b3-9405-2e
4367f5f4-69a2-4938-adff-aa7b1ea0ef3d 0      false      []      0      {}      {}      {}      false      [0f9cc16e-39a2-4707-bee4-49
1d8a99af05] []      "s2-eth2" {}      []      false      []      0      {}      {}      {}      false      [52cc0226-31b3-4874-8930-c2
b2ee38e5-429e-42d6-af64-17cd455ccf7f 0      false      []      0      {}      {}      {}      false      [7472bdbc-1230-4adf-b863-47
c3f765e98b] []      "s2-eth3" {}      []      false      []      0      {}      {}      {}      false      [25aa243b-006d-4475-8a50-eb
171d21f5-04c7-4719-bce1-74773c3bf1e5 0      false      []      0      {}      {}      {}      false      [208a1ff4-71f1-48b3-9405-2e
c5a5f0ec30] []      "s3"      {}      []      false      []      0      {}      {}      {}      false      [25aa243b-006d-4475-8a50-eb
e8954ce9-5f79-4a23-b4bb-e7dbbe3aa748 0      false      []      0      {}      {}      {}      false      [208a1ff4-71f1-48b3-9405-2e
9021df271a] []      "s3-eth1" {}      []      false      []      0      {}      {}      {}      false      [208a1ff4-71f1-48b3-9405-2e
5a5fd768-0d94-43cf-b81b-ff9a84b25f5b 0      false      []      0      {}      {}      {}      false      [208a1ff4-71f1-48b3-9405-2e
7691804e53] []      "s3-eth2" {}      []      false      []      0      {}      {}      {}      false      [208a1ff4-71f1-48b3-9405-2e

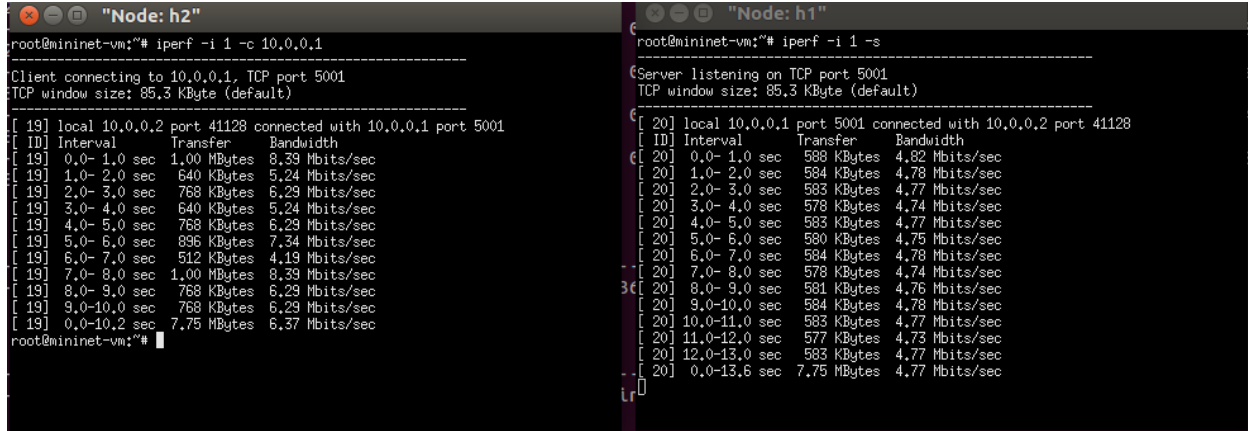
QoS table
_uuid      external_ids other_config      queues      type
-----
37115d01-61aa-4609-a2bf-7834d3dcec8b {}      {max-rate="5000000"} {1=b5f36cc5-fcf9-4272-bfcb-0b3f1965b6d3} linux-htb

Queue table
_uuid      dscp external_ids other_config
-----
b5f36cc5-fcf9-4272-bfcb-0b3f1965b6d3 []      {}      {max-rate="5000000", min-rate="500000"}

SSL table
_uuid bootstrap_ca_cert ca_cert certificate external_ids private_key
-----

sFlow table
_uuid agent external_ids header polling sampling targets
-----
mininet@mininet-vm:~$
```

Result for Iperf server and client:



```
Node: h2
root@mininet-vm:~# iperf -i 1 -c 10.0.0.1
Client connecting to 10.0.0.1, TCP port 5001
TCP window size: 85.3 KByte (default)
[ 19] local 10.0.0.2 port 41128 connected with 10.0.0.1 port 5001
[ ID] Interval      Transfer      Bandwidth
[ 19] 0.0- 1.0 sec  1.00 MBytes  8.33 Mbits/sec
[ 19] 1.0- 2.0 sec  640 KBytes  5.24 Mbits/sec
[ 19] 2.0- 3.0 sec  768 KBytes  6.29 Mbits/sec
[ 19] 3.0- 4.0 sec  640 KBytes  5.24 Mbits/sec
[ 19] 4.0- 5.0 sec  768 KBytes  6.29 Mbits/sec
[ 19] 5.0- 6.0 sec  896 KBytes  7.34 Mbits/sec
[ 19] 6.0- 7.0 sec  512 KBytes  4.19 Mbits/sec
[ 19] 7.0- 8.0 sec  1.00 MBytes  8.39 Mbits/sec
[ 19] 8.0- 9.0 sec  768 KBytes  6.29 Mbits/sec
[ 19] 9.0-10.0 sec  768 KBytes  6.29 Mbits/sec
[ 19] 0.0-10.2 sec  7.75 MBytes  6.37 Mbits/sec
root@mininet-vm:~#

Node: h1
root@mininet-vm:~# iperf -i 1 -s
Server listening on TCP port 5001
TCP window size: 85.3 KByte (default)
[ 20] local 10.0.0.1 port 5001 connected with 10.0.0.2 port 41128
[ ID] Interval      Transfer      Bandwidth
[ 20] 0.0- 1.0 sec  588 KBytes  4.82 Mbits/sec
[ 20] 1.0- 2.0 sec  584 KBytes  4.78 Mbits/sec
[ 20] 2.0- 3.0 sec  583 KBytes  4.77 Mbits/sec
[ 20] 3.0- 4.0 sec  578 KBytes  4.74 Mbits/sec
[ 20] 4.0- 5.0 sec  583 KBytes  4.77 Mbits/sec
[ 20] 5.0- 6.0 sec  580 KBytes  4.75 Mbits/sec
[ 20] 6.0- 7.0 sec  584 KBytes  4.78 Mbits/sec
[ 20] 7.0- 8.0 sec  578 KBytes  4.74 Mbits/sec
[ 20] 8.0- 9.0 sec  581 KBytes  4.76 Mbits/sec
[ 20] 9.0-10.0 sec  584 KBytes  4.78 Mbits/sec
[ 20] 10.0-11.0 sec  583 KBytes  4.77 Mbits/sec
[ 20] 11.0-12.0 sec  577 KBytes  4.73 Mbits/sec
[ 20] 12.0-13.0 sec  583 KBytes  4.77 Mbits/sec
[ 20] 0.0-13.6 sec  7.75 MBytes  4.77 Mbits/sec
```

Results show that traffic below 5mbps is only received by Server h1 even though client generates traffic above 5mbps.

Hence, we can see that we have manually shaped the traffic by controlling the bandwidth as shown in above screenshots.

APPROACH 2: Shaping the traffic using Restconf APIs i.e over application layer with the help of postman:

A . Connect to OVSDB server using passive connection:

```
mininet@mininet-vm:~$ sudo ovs-vsctl set-manager tcp:172.16.75.28:6640
mininet@mininet-vm:~$ sudo ovs-vsctl show
918037ec-b307-45d7-a75a-f1ac4337d135
    Manager "tcp:172.16.75.28:6640"
    is_connected: true
```

For the following examples, the following OVS host is configured. We have configured OVSDB host as HOST1 and this host is used further for interacting with controller for various operations :

```
http:// <controller-ip> :8181/restconf/config/network-topology:network-
topology/topology/ovsdb:1/
```

Body:

```
{
  "node": [
    {
      "node-id": "ovsdb:HOST1",
      "connection-info": {
```

```

        "ovsdb:remote-ip": "<ovs-host-ip>"
        "ovsdb:remote-port": "<ovs-host-ovsdb-port>"
    }
}
]
}

```

Where

- *<controller-ip>* = localhost is the IP address of the OpenDaylight controller
- *<ovs-host-ip>* = 172.16.75.28 is the IP address of the OVS host
- *<ovs-host-ovsdb-port>* = 46706 is the TCP port of the OVSDB server on the OVS host (e.g. 6640)

If all of the previous steps were successful, a query of the operational MD-SAL should look something like the following results. This indicates that the configuration commands have been successfully instantiated on the OVS host.

HTTP GET:

```

http://localhost:8181/restconf/operational/network-topology:network-
topology/topology/ovsdb:1/

```

RESULT BODY:

```

{
  "topology": [
    {
      "topology-id": "ovsdb:1",
      "node": [
        {
          "node-id": "ovsdb:HOST1",
          "ovsdb:db-version": "7.3.0",
          "ovsdb:connection-info": {
            "remote-port": 46706,
            "local-ip": "172.16.75.28",
            "local-port": 6640,
            "remote-ip": "172.16.75.28"
          }
        }
      ]
    }
  ]
}

```

B. Create a row in Queue and set given Bandwidth:

Create a new Queue in the configuration MD-SAL.

HTTP PUT:

```
http://localhost:8181/restconf/config/network-topology:network-topology/topology/ovsdb:1/node/ovsdb:HOST1/ovsdb:queues/QUEUE-1/
```

Body

```
{
  "ovsdb:queues": [
    {
      "queue-id": "QUEUE-1",
      "dscp": 25,
      "queues-other-config": [
        {
          "queue-other-config-key": "max-rate",
          "queue-other-config-value": "5000000"
        }
      ]
    }
  ]
}
```

```
Queue table
 _uuid          dscp external_ids          other_config
-----
c01a66a0-635b-4459-ba5e-c09cb2a0473e 25 {opendaylight-iid="/network-topology:network-topology/network-topology:topology[network-topology:topology-id='ovsdb:1']/network-topology:node[network-topology:node-id='ovsdb:HOST1']/ovsdb:queues[ovsdb:queue-id='QUEUE-1']"} {max-rate="5000000"}
```

C. Create QoS:

Create a QoS entry. Note that the UUID of the Queue entry, obtained by querying the operational MD-SAL of the Queue entry, is specified in the queue-list of the QoS entry. Queue entries may be added to the QoS entry at the creation of the QoS entry, or by a subsequent update to the QoS entry.

HTTP PUT:

```
http://localhost:8181/restconf/config/network-topology:network-topology/topology/ovsdb:1/node/ovsdb:HOST1/ovsdb:qos-entries/QOS-1/
```

BODY :

```
{
  "ovsdb:qos-entries": [
    {
      "qos-id": "QOS-1",
      "qos-type": "ovsdb:qos-type-linux-htb",
      "qos-other-config": [
        {
          "other-config-key": "max-rate",
          "other-config-value": "5000000"
        }
      ],
      "queue-list": [
        {
          "queue-number": "0",
          "queue-uuid": "83640357-3596-4877-9527-b472aa854d69"
        }
      ]
    }
  ]
}
```

D. Add QoS to a Port

Update the termination point entry to include the UUID of the QoS entry, obtained by querying the operational MD-SAL, to associate a QoS entry with a port.

HTTP PUT:

```
http://localhost:8181/restconf/config/network-topology:network-topology/topology/ovsdb:1/node/ovsdb:HOST1%2Fbridge%2Fbr-test/termination-point/testport/
```

Body:

```
{
  "network-topology:termination-point": [
    {
      "ovsdb:name": "testport",
      "tp-id": "testport",
      "qos": "90ba9c60-3aac-499d-9be7-555f19a6bb31"
    }
  ]
}
```

Results in Iperf Server and Client.

```
Node: h2
root@mininet-vm:~# iperf -i 1 -c 10.0.0.1
Client connecting to 10.0.0.1, TCP port 5001
TCP window size: 85.3 KByte (default)
[ 19] local 10.0.0.2 port 41128 connected with 10.0.0.1 port 5001
[ ID] Interval      Transfer      Bandwidth
[ 19] 0.0- 1.0 sec  1.00 MBytes  8.39 Mbits/sec
[ 19] 1.0- 2.0 sec  640 KBytes  5.24 Mbits/sec
[ 19] 2.0- 3.0 sec  768 KBytes  6.29 Mbits/sec
[ 19] 3.0- 4.0 sec  640 KBytes  5.24 Mbits/sec
[ 19] 4.0- 5.0 sec  768 KBytes  6.29 Mbits/sec
[ 19] 5.0- 6.0 sec  896 KBytes  7.34 Mbits/sec
[ 19] 6.0- 7.0 sec  512 KBytes  4.19 Mbits/sec
[ 19] 7.0- 8.0 sec  1.00 MBytes  8.39 Mbits/sec
[ 19] 8.0- 9.0 sec  768 KBytes  6.29 Mbits/sec
[ 19] 9.0-10.0 sec  768 KBytes  6.29 Mbits/sec
[ 19] 10.0-11.0 sec 7.75 MBytes  6.37 Mbits/sec
root@mininet-vm:~#

Node: h1
root@mininet-vm:~# iperf -i 1 -s
Server listening on TCP port 5001
TCP window size: 85.3 KByte (default)
[ 20] local 10.0.0.1 port 5001 connected with 10.0.0.2 port 41128
[ ID] Interval      Transfer      Bandwidth
[ 20] 0.0- 1.0 sec  588 KBytes  4.82 Mbits/sec
[ 20] 1.0- 2.0 sec  584 KBytes  4.78 Mbits/sec
[ 20] 2.0- 3.0 sec  583 KBytes  4.77 Mbits/sec
[ 20] 3.0- 4.0 sec  578 KBytes  4.74 Mbits/sec
[ 20] 4.0- 5.0 sec  583 KBytes  4.77 Mbits/sec
[ 20] 5.0- 6.0 sec  580 KBytes  4.75 Mbits/sec
[ 20] 6.0- 7.0 sec  584 KBytes  4.78 Mbits/sec
[ 20] 7.0- 8.0 sec  578 KBytes  4.74 Mbits/sec
[ 20] 8.0- 9.0 sec  581 KBytes  4.76 Mbits/sec
[ 20] 9.0-10.0 sec  584 KBytes  4.78 Mbits/sec
[ 20] 10.0-11.0 sec  583 KBytes  4.77 Mbits/sec
[ 20] 11.0-12.0 sec  577 KBytes  4.73 Mbits/sec
[ 20] 12.0-13.0 sec  583 KBytes  4.77 Mbits/sec
[ 20] 0.0-13.0 sec  7.75 MBytes  4.77 Mbits/sec
```


6. Conclusion

We studied Software Defined Networks thoroughly. We are able to understand and set up a topology using mininet and OpenDaylight controller. We have successfully generated the traffic and shaped it according to our requirement using manual commands as well as Restconf API provided by OpenDayLight.

7. References:

- [1] <http://www.brianlinkletter.com/set-up-mininet/>
- [2] <http://www.brianlinkletter.com/using-the-openswitch-sdn-controller-with-the-mininet-network-emulator/>
- [3] Reference book by Tiwari Vivek.-SDN and Openflow for beginners with hands on labs
- [4] <http://docs.openvswitch.org/en/stable-nitrogen/user-guide/ovsdb-user-guide.html#modelling-of-qos-and-queue-tables-in-openvswitch-md-sal>

