# Implementation of Queue Configuration and Bandwidth control using Ryu controller

## Test Environment:

SDN Hub provides a VM which has inbuilt SDN controllers like POX, Ryu, Floodlight, etc. Make sure you update the Ryu controller provided by SDN hub or remove the existing Ryu controller and clone the latest Ryu repository from the GitHub with the steps as given below:

## SDN HUB URL:

http://sdnhub.org/releases/sdn-starter-kit-ryu/

## Steps for installing Komodo-Edit on Linux Machine: (One of the good editors for editing and working on the codes)

```
$ sudo add-apt-repository ppa:mystic-mirage/komodo-edit
$ sudo apt-get update
$ sudo apt-get install komodo-edit
```

## Steps for installing the latest Ryu SDN controller:

```
$ git clone git://github.com/osrg/ryu.git
$ cd ryu; python ./setup.py install
```

## Steps for installing the Ryu related dependencies in order to use optional functionalities like OF-config:

```
$ cd ryu
$ sudo pip install -r tools/pip-requires
$ sudo python setup.py install
```

## Steps for creating a module in Ryu and which is supposed to be processed on Flow Table pipeline processing, we need to modify simple_switch_13.py to register flow entry into table id:1 as shown below:

```
$ sed '/OFPFlowMod(/,/)/s/)/, table_id=1)/' ryu/ryu/app/simple_switch_13.py >
ryu/ryu/app/perf.py

$ cd ryu/; python ./setup.py install
```

**Note:** If you change anything in the created module, then again rebuild the Ryu controller to observe the changes, otherwise it won’t take the changes by using the command below:

```
$ cd ryu/; python ./setup.py install
```

## Steps for running the Bandwidth control and Queue management module using mininet based topology and Ryu controller:

**Step 1:** Run the Ryu controller with the created Bandwidth control and Queue management module (perf.py), rest_qos.py and rest_conf_switch.py by using the command as shown below:

```
$ ryu-manager ryu.app.rest_qos ryu.app.perf ryu.app.rest_conf_switch
```

-----------------------------------------------------------------------------------------------------------------

**Step 2:** Run the created custom mininet topology using the command as shown below:

```
$ sudo mn --custom test_topo.py --topo mytopo --switch ovsk --controller remote –mac
```

Note: Once the mininet topology is created do pingall on mininet console.

-----------------------------------------------------------------------------------------------------------------

**Step 3:** Check for the installed Queues on a particular switch (in this case it's on switch-2) by using the below mentioned url in the browser:

http://localhost:8080/qos/queue/0000000000000002

Note: You can also use curl with GET to check the Queues installed on a switch by using the command from the command prompt as below:

```
$ curl -X GET http://localhost:8080/qos/queue/0000000000000002
```

-----------------------------------------------------------------------------------------------------------------

**Step 4:** Now open the console for the hosts h1, h2, h3 and h3 for testing the performance and the provided bandwidth by using the command

```
Mininet > xterm h1 h3
```

-----------------------------------------------------------------------------------------------------------------

**Step 5:** Measure the bandwidth by using iperf as shown below:

On h3 console use the command $iperf  -s  -u -i 1 -p 5001

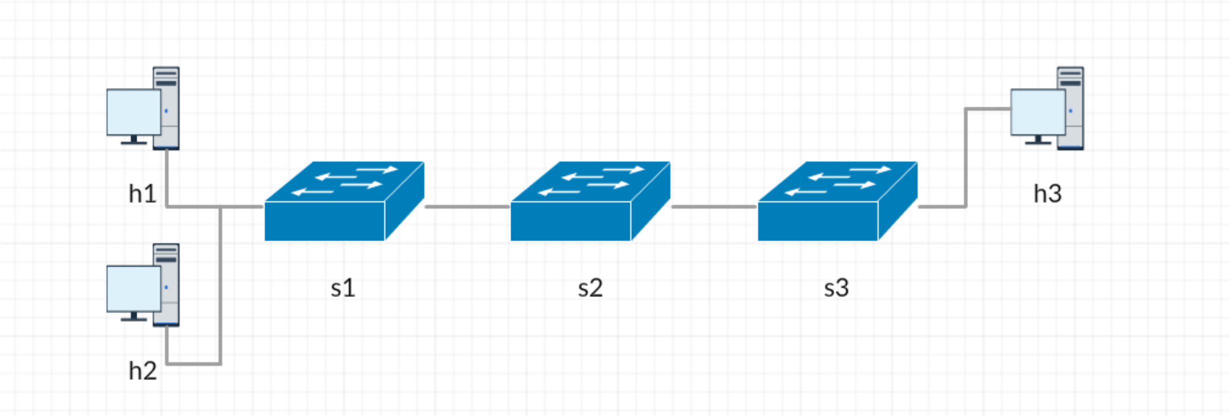
On h1 console use the command $iperf -c 10.0.0.3 -u -b 1M -p 5001

The above commands simply mean that we are testing the bandwidth for the UDP packets that are transmitted at an interval of 1 sec and where h3 is acting as a server and h1 is acting as a client.

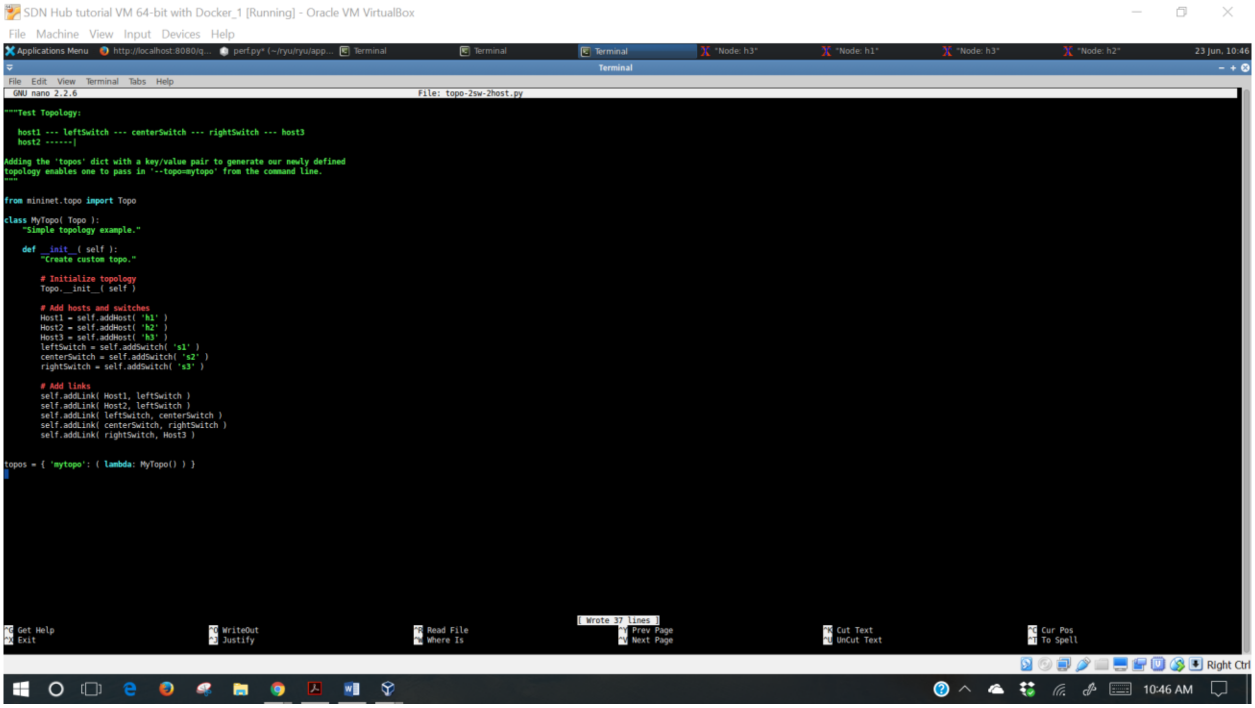
-----------------------------------------------------------------------------------------------------------------

Test Cases:

For testing the bandwidth control and Queue management on switches, we have created a simple test topology with three hosts and three switches.
In this topology, host (h1) and host (h2) are connected to switch (s1), switch s1 is connected to switch (s2), switch (s2) is connected to switch (s3) and host (h3) is connected to switch (s3) as depicted in the figure below:



Test Topology using Mininet:

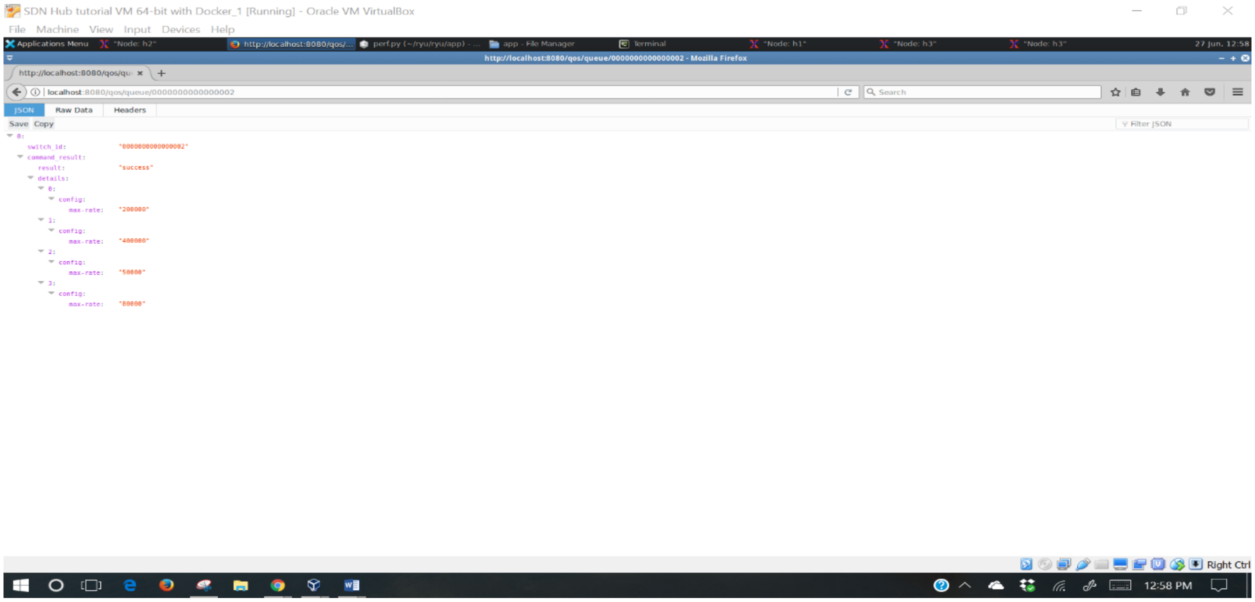

SDN Hub tutorial VM 64-bit with Docker_1 [Running] - Oracle VM VirtualBox
File Machine View Input Devices Help
Applications Menu
http://localhost:8080/q...
perf.py* (~/ryu/ryu/app...
Terminal
Terminal
Terminal
*Node: h3*
*Node: h1*
*Node: h3*
*Node: h2*
23 Jun, 10:46
Terminal
File Edit View Terminal Tabs Help
Right Ctrl
10:46 AM

```
GNU nano 2.2.6                                  File: topo-2sw-2host.py

"""Test Topology:

   host1 --- leftSwitch --- centerSwitch --- rightSwitch --- host3
   host2 ------|

Adding the 'topos' dict with a key/value pair to generate our newly defined
topology enables one to pass in '--topo=mytopo' from the command line.
"""

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
        Host1 = self.addHost( 'h1' )
        Host2 = self.addHost( 'h2' )
        Host3 = self.addHost( 'h3' )
        leftSwitch = self.addSwitch( 's1' )
        centerSwitch = self.addSwitch( 's2' )
        rightSwitch = self.addSwitch( 's3' )

        # Add links
        self.addLink( Host1, leftSwitch )
        self.addLink( Host2, leftSwitch )
        self.addLink( leftSwitch, centerSwitch )
        self.addLink( centerSwitch, rightSwitch )
        self.addLink( rightSwitch, Host3 )


topos = { 'mytopo': ( lambda: MyTopo() ) }
```

```
                                       [ Wrote 37 lines ]
^G Get Help      ^O WriteOut      ^R Read File     ^Y Prev Page     ^K Cut Text      ^C Cur Pos
^X Exit          ^J Justify       ^W Where Is      ^V Next Page     ^U UnCut Text    ^T To Spell
```

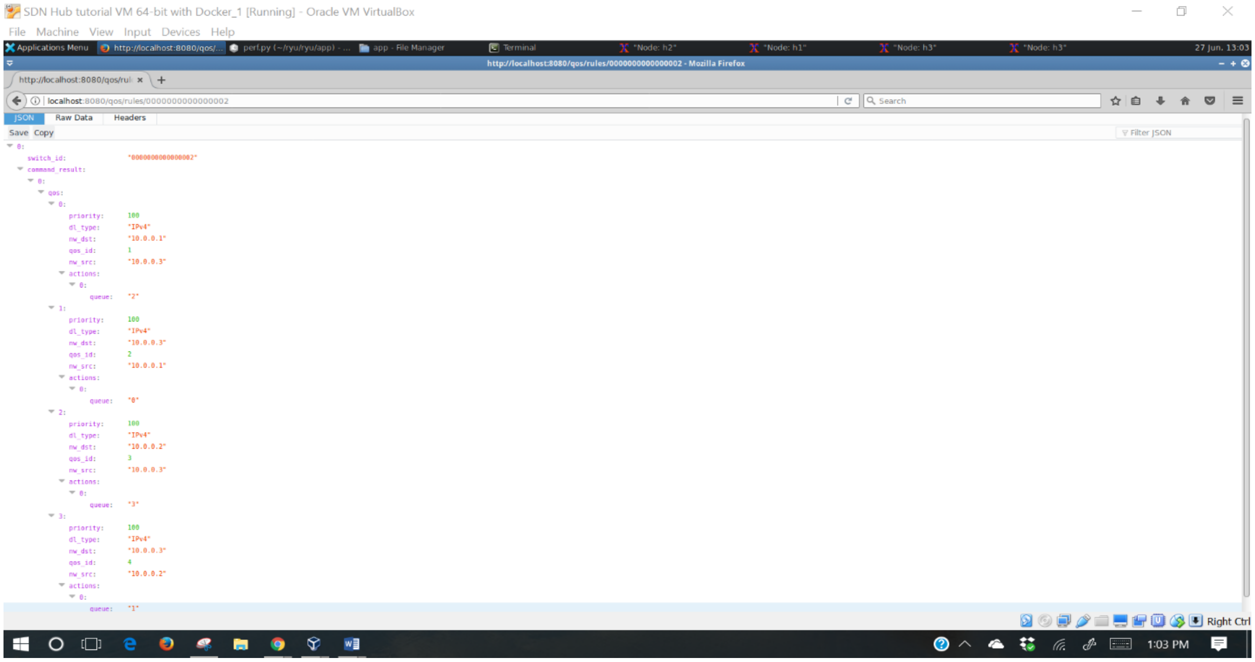
## Test Scenario 1:

In this test scenario two queues (q0 and q1) with maximum rates as 200 Kbps and 400 Kbps are configured to switch-2 port s2-eth1 and two queues (q2 and q3) with maximum rates as 50 Kbps and 80 Kbps are configured to switch-2 port s2-eth2 respectively.


SDN Hub tutorial VM 64-bit with Docker_1 [Running] - Oracle VM VirtualBox
File Machine View Input Devices Help
Applications Menu
"Node: h2"
http://localhost:8080/qos/...
perf.py (~/ryu/ryu/app) - ...
app - File Manager
Terminal
"Node: h1"
"Node: h3"
"Node: h3"
27 Jun, 12:58
http://localhost:8080/qos/queue/0000000000000002 - Mozilla Firefox
http://localhost:8080/qos/qu
localhost:8080/qos/queue/0000000000000002
Search
JSON
Raw Data
Headers
Save
Copy
Filter JSON
Right Ctrl
12:58 PM

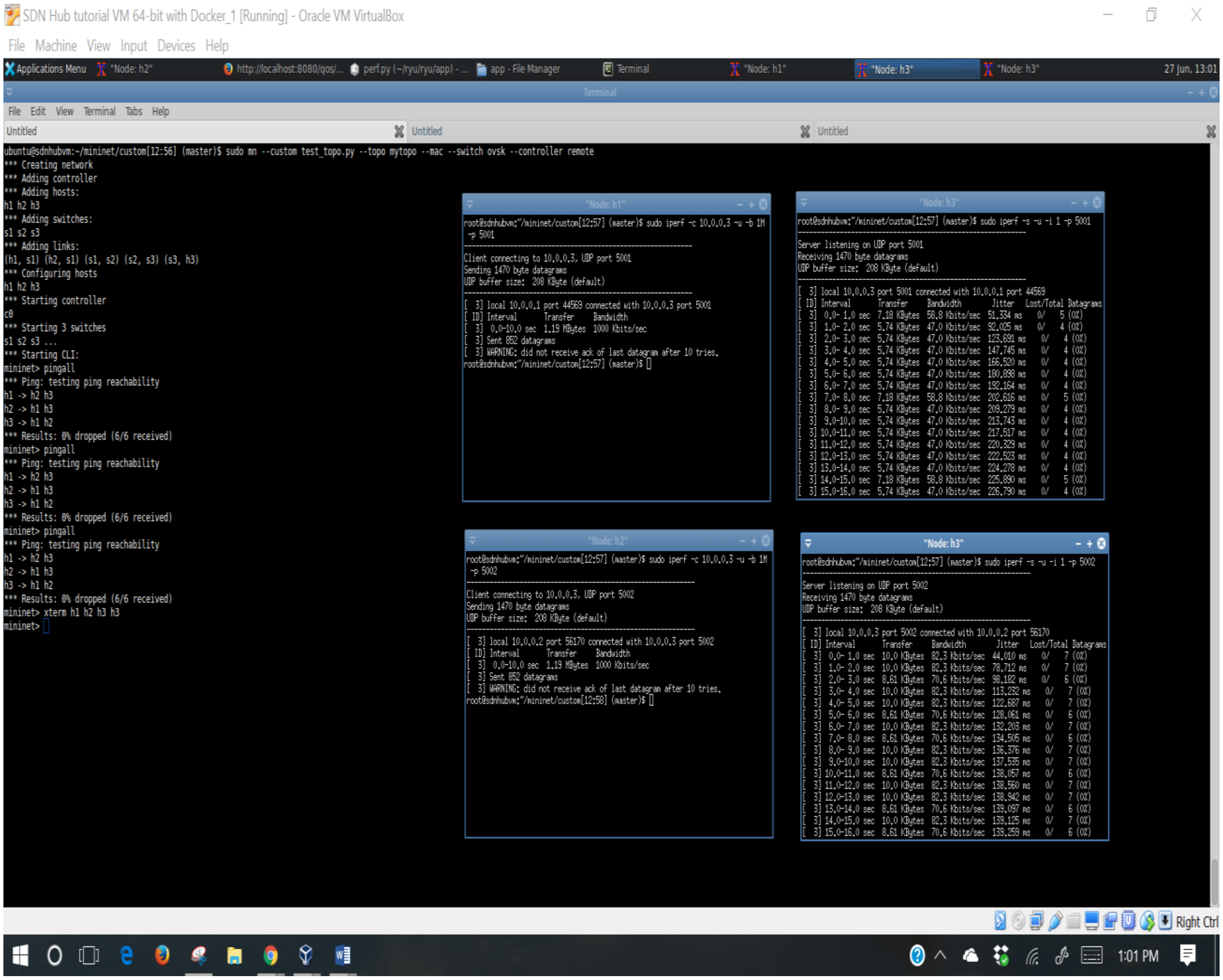
```
▼ 0:
    switch_id:          "0000000000000002"
  ▼ command_result:
      result:           "success"
    ▼ details:
      ▼ 0:
        ▼ config:
            max-rate:   "200000"
      ▼ 1:
        ▼ config:
            max-rate:   "400000"
      ▼ 2:
        ▼ config:
            max-rate:   "50000"
      ▼ 3:
        ▼ config:
            max-rate:   "80000"
```

Packets from 10.0.0.1 (h1) to 10.0.0.3 (h3) are routed through the queue (q0) on s2-eth1 and the packets from 10.0.0.3 (h3) to 10.0.0.1 (h1) are routed through queue the (q2) on s2-eth2 whereas, the packets from 10.0.0.2 (h2) to 10.0.0.3 (h3) are routed through the queue (q1) on s2-eth1 and the packets from 10.0.0.3 (h3) to 10.0.0.2 (h2) are routed through the queue (q3) on s2-eth2.


SDN Hub tutorial VM 64-bit with Docker_1 [Running] - Oracle VM VirtualBox
File Machine View Input Devices Help
Applications Menu
http://localhost:8080/qos/...
perf.py (~/ryu/ryu/app) - ...
app - File Manager
Terminal
"Node: h2"
"Node: h1"
"Node: h3"
"Node: h3"
27 Jun, 13:03
http://localhost:8080/qos/rules/0000000000000002 - Mozilla Firefox
http://localhost:8080/qos/rul
localhost:8080/qos/rules/0000000000000002
Search
JSON
Raw Data
Headers
Save
Copy
Filter JSON
Right Ctrl
1:03 PM

```
▼ 0:
    switch_id:            "0000000000000002"
  ▼ command_result:
    ▼ 0:
      ▼ qos:
        ▼ 0:
            priority:     100
            dl_type:      "IPv4"
            nw_dst:       "10.0.0.1"
            qos_id:       1
            nw_src:       "10.0.0.3"
          ▼ actions:
            ▼ 0:
                queue:    "2"
        ▼ 1:
            priority:     100
            dl_type:      "IPv4"
            nw_dst:       "10.0.0.3"
            qos_id:       2
            nw_src:       "10.0.0.1"
          ▼ actions:
            ▼ 0:
                queue:    "0"
        ▼ 2:
            priority:     100
            dl_type:      "IPv4"
            nw_dst:       "10.0.0.2"
            qos_id:       3
            nw_src:       "10.0.0.3"
          ▼ actions:
            ▼ 0:
                queue:    "3"
        ▼ 3:
            priority:     100
            dl_type:      "IPv4"
            nw_dst:       "10.0.0.3"
            qos_id:       4
            nw_src:       "10.0.0.2"
          ▼ actions:
            ▼ 0:
                queue:    "1"
```

We have measured the bandwidth by using iperf, where h3(server) listens on the port 5001 and port 5002. h1(client) and h2(client) sends 1Mbps UDP traffic to the port 5001 on h3 and 1Mbps UDP traffic to the port 5002 on h3.


SDN Hub tutorial VM 64-bit with Docker_1 [Running] - Oracle VM VirtualBox
File Machine View Input Devices Help
Applications Menu
"Node: h2"
http://localhost:8080/qos/...
perf.py (~/ryu/ryu/app) - ...
app - File Manager
Terminal
"Node: h1"
"Node: h3"
"Node: h3"
27 Jun, 13:01
Terminal
File Edit View Terminal Tabs Help
Untitled
Untitled
Untitled
Right Ctrl
1:01 PM

```
ubuntu@sdnhubvm:~/mininet/custom[12:56] (master)$ sudo mn --custom test_topo.py --topo mytopo --mac --switch ovsk --controller remote
*** Creating network
*** Adding controller
*** Adding hosts:
h1 h2 h3
*** Adding switches:
s1 s2 s3
*** Adding links:
(h1, s1) (h2, s1) (s1, s2) (s2, s3) (s3, h3)
*** Configuring hosts
h1 h2 h3
*** Starting controller
c0
*** Starting 3 switches
s1 s2 s3 ...
*** Starting CLI:
mininet> pingall
*** Ping: testing ping reachability
h1 -> h2 h3
h2 -> h1 h3
h3 -> h1 h2
*** Results: 0% dropped (6/6 received)
mininet> pingall
*** Ping: testing ping reachability
h1 -> h2 h3
h2 -> h1 h3
h3 -> h1 h2
*** Results: 0% dropped (6/6 received)
mininet> pingall
*** Ping: testing ping reachability
h1 -> h2 h3
h2 -> h1 h3
h3 -> h1 h2
*** Results: 0% dropped (6/6 received)
mininet> xterm h1 h2 h3 h3
mininet>
```

"Node: h1"

```
root@sdnhubvm:~/mininet/custom[12:57] (master)$ sudo iperf -c 10.0.0.3 -u -b 1M
 -p 5001
------------------------------------------------------------
Client connecting to 10.0.0.3, UDP port 5001
Sending 1470 byte datagrams
UDP buffer size:  208 KByte (default)
------------------------------------------------------------
[  3] local 10.0.0.1 port 44569 connected with 10.0.0.3 port 5001
[ ID] Interval       Transfer     Bandwidth
[  3]  0.0-10.0 sec  1.19 MBytes  1000 Kbits/sec
[  3] Sent 852 datagrams
[  3] WARNING: did not receive ack of last datagram after 10 tries.
root@sdnhubvm:~/mininet/custom[12:57] (master)$
```

"Node: h3"

```
root@sdnhubvm:~/mininet/custom[12:57] (master)$ sudo iperf -s -u -i 1 -p 5001
------------------------------------------------------------
Server listening on UDP port 5001
Receiving 1470 byte datagrams
UDP buffer size:  208 KByte (default)
------------------------------------------------------------
[  3] local 10.0.0.3 port 5001 connected with 10.0.0.1 port 44569
[ ID] Interval       Transfer     Bandwidth        Jitter   Lost/Total Datagrams
[  3]  0.0- 1.0 sec  7.18 KBytes  58.8 Kbits/sec  51.334 ms    0/    5 (0%)
[  3]  1.0- 2.0 sec  5.74 KBytes  47.0 Kbits/sec  92.025 ms    0/    4 (0%)
[  3]  2.0- 3.0 sec  5.74 KBytes  47.0 Kbits/sec  123.691 ms    0/    4 (0%)
[  3]  3.0- 4.0 sec  5.74 KBytes  47.0 Kbits/sec  147.745 ms    0/    4 (0%)
[  3]  4.0- 5.0 sec  5.74 KBytes  47.0 Kbits/sec  166.520 ms    0/    4 (0%)
[  3]  5.0- 6.0 sec  5.74 KBytes  47.0 Kbits/sec  180.898 ms    0/    4 (0%)
[  3]  6.0- 7.0 sec  5.74 KBytes  47.0 Kbits/sec  192.164 ms    0/    4 (0%)
[  3]  7.0- 8.0 sec  7.18 KBytes  58.8 Kbits/sec  202.616 ms    0/    5 (0%)
[  3]  8.0- 9.0 sec  5.74 KBytes  47.0 Kbits/sec  209.279 ms    0/    4 (0%)
[  3]  9.0-10.0 sec  5.74 KBytes  47.0 Kbits/sec  213.743 ms    0/    4 (0%)
[  3] 10.0-11.0 sec  5.74 KBytes  47.0 Kbits/sec  217.517 ms    0/    4 (0%)
[  3] 11.0-12.0 sec  5.74 KBytes  47.0 Kbits/sec  220.329 ms    0/    4 (0%)
[  3] 12.0-13.0 sec  5.74 KBytes  47.0 Kbits/sec  222.523 ms    0/    4 (0%)
[  3] 13.0-14.0 sec  5.74 KBytes  47.0 Kbits/sec  224.278 ms    0/    4 (0%)
[  3] 14.0-15.0 sec  7.18 KBytes  58.8 Kbits/sec  225.890 ms    0/    5 (0%)
[  3] 15.0-16.0 sec  5.74 KBytes  47.0 Kbits/sec  226.790 ms    0/    4 (0%)
```

"Node: h2"

```
root@sdnhubvm:~/mininet/custom[12:57] (master)$ sudo iperf -c 10.0.0.3 -u -b 1M
 -p 5002
------------------------------------------------------------
Client connecting to 10.0.0.3, UDP port 5002
Sending 1470 byte datagrams
UDP buffer size:  208 KByte (default)
------------------------------------------------------------
[  3] local 10.0.0.2 port 56170 connected with 10.0.0.3 port 5002
[ ID] Interval       Transfer     Bandwidth
[  3]  0.0-10.0 sec  1.19 MBytes  1000 Kbits/sec
[  3] Sent 852 datagrams
[  3] WARNING: did not receive ack of last datagram after 10 tries.
root@sdnhubvm:~/mininet/custom[12:58] (master)$
```

"Node: h3"

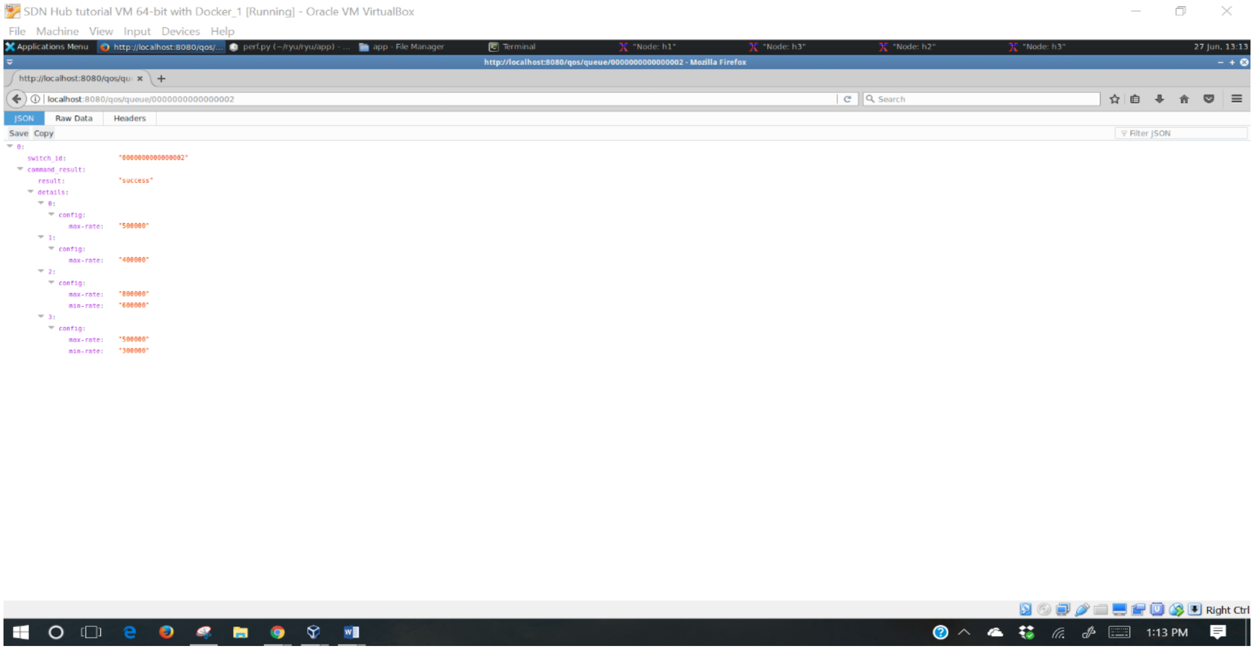
```
root@sdnhubvm:~/mininet/custom[12:57] (master)$ sudo iperf -s -u -i 1 -p 5002
------------------------------------------------------------
Server listening on UDP port 5002
Receiving 1470 byte datagrams
UDP buffer size:  208 KByte (default)
------------------------------------------------------------
[  3] local 10.0.0.3 port 5002 connected with 10.0.0.2 port 56170
[ ID] Interval       Transfer     Bandwidth        Jitter   Lost/Total Datagrams
[  3]  0.0- 1.0 sec  10.0 KBytes  82.3 Kbits/sec  44.010 ms    0/    7 (0%)
[  3]  1.0- 2.0 sec  10.0 KBytes  82.3 Kbits/sec  78.712 ms    0/    7 (0%)
[  3]  2.0- 3.0 sec  8.61 KBytes  70.6 Kbits/sec  98.182 ms    0/    6 (0%)
[  3]  3.0- 4.0 sec  10.0 KBytes  82.3 Kbits/sec  113.232 ms    0/    7 (0%)
[  3]  4.0- 5.0 sec  10.0 KBytes  82.3 Kbits/sec  122.687 ms    0/    7 (0%)
[  3]  5.0- 6.0 sec  8.61 KBytes  70.6 Kbits/sec  128.061 ms    0/    6 (0%)
[  3]  6.0- 7.0 sec  10.0 KBytes  82.3 Kbits/sec  132.203 ms    0/    7 (0%)
[  3]  7.0- 8.0 sec  8.61 KBytes  70.6 Kbits/sec  134.505 ms    0/    6 (0%)
[  3]  8.0- 9.0 sec  10.0 KBytes  82.3 Kbits/sec  136.376 ms    0/    7 (0%)
[  3]  9.0-10.0 sec  10.0 KBytes  82.3 Kbits/sec  137.535 ms    0/    7 (0%)
[  3] 10.0-11.0 sec  8.61 KBytes  70.6 Kbits/sec  138.057 ms    0/    6 (0%)
[  3] 11.0-12.0 sec  10.0 KBytes  82.3 Kbits/sec  138.560 ms    0/    7 (0%)
[  3] 12.0-13.0 sec  10.0 KBytes  82.3 Kbits/sec  138.942 ms    0/    7 (0%)
[  3] 13.0-14.0 sec  8.61 KBytes  70.6 Kbits/sec  139.097 ms    0/    6 (0%)
[  3] 14.0-15.0 sec  10.0 KBytes  82.3 Kbits/sec  139.125 ms    0/    7 (0%)
[  3] 15.0-16.0 sec  8.61 KBytes  70.6 Kbits/sec  139.259 ms    0/    6 (0%)
```

The above figure shows the implementation results for the Queue management and Bandwidth control using Ryu for the Test scenario 1.

The above result shows that, traffic sent to the port 5001(h1 and h3) is shaped with up to 50Kbps and the traffic to the port 5002 (h2 and h3) is shaped with up to 80Kbps bandwidth.

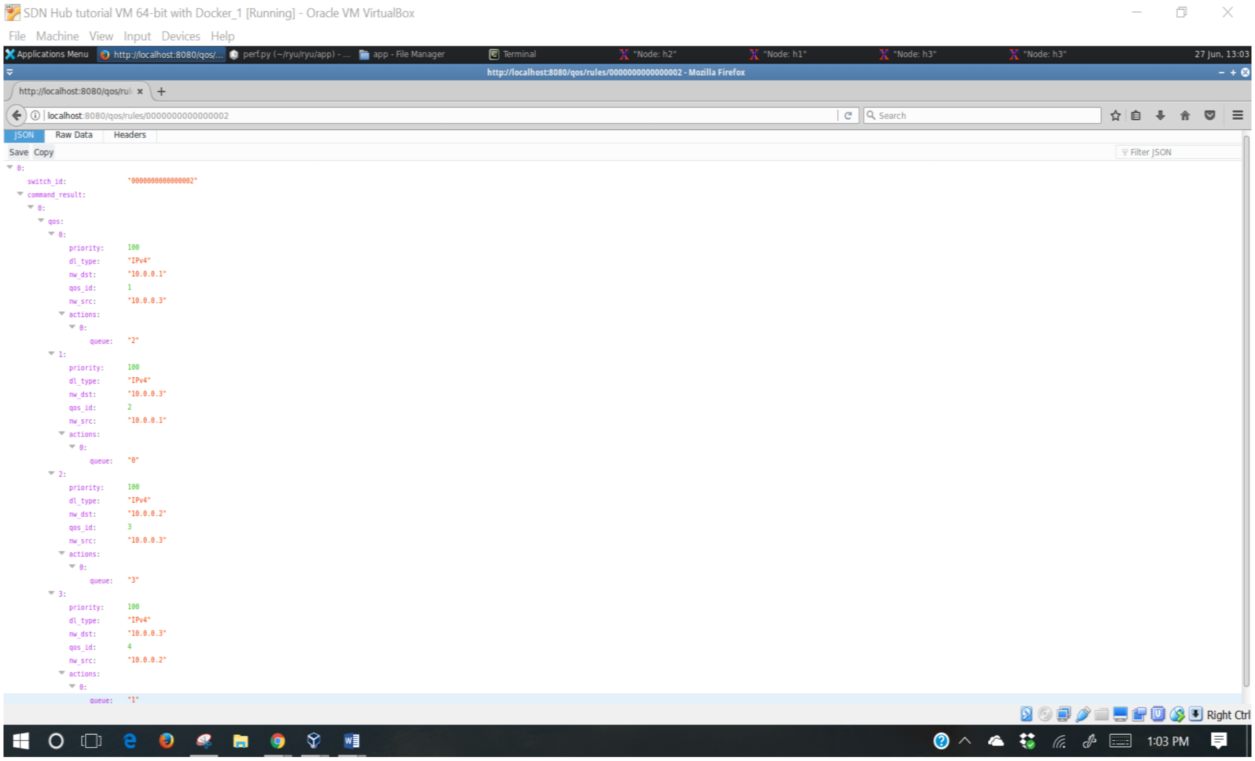
## Test Scenario 2:

In this test scenario two queues (q0 and q1) with maximum rates as 500 Kbps and 400 Kbps are configured to switch-2 port s2-eth1 and two queues (q2 and q3) with maximum and minimum rates as [800 Kbps (max) & 600 Kbps (min)] and [500 Kbps (max) & 300 (min)] are configured to switch-2 port s2-eth2 respectively.


SDN Hub tutorial VM 64-bit with Docker_1 [Running] - Oracle VM VirtualBox
File Machine View Input Devices Help
Applications Menu
http://localhost:8080/qos/queue/0000000000000002 - Mozilla Firefox
localhost:8080/qos/queue/0000000000000002
JSON
Raw Data
Headers
Save
Copy
Filter JSON
27 Jun, 13:13
Right Ctrl
1:13 PM

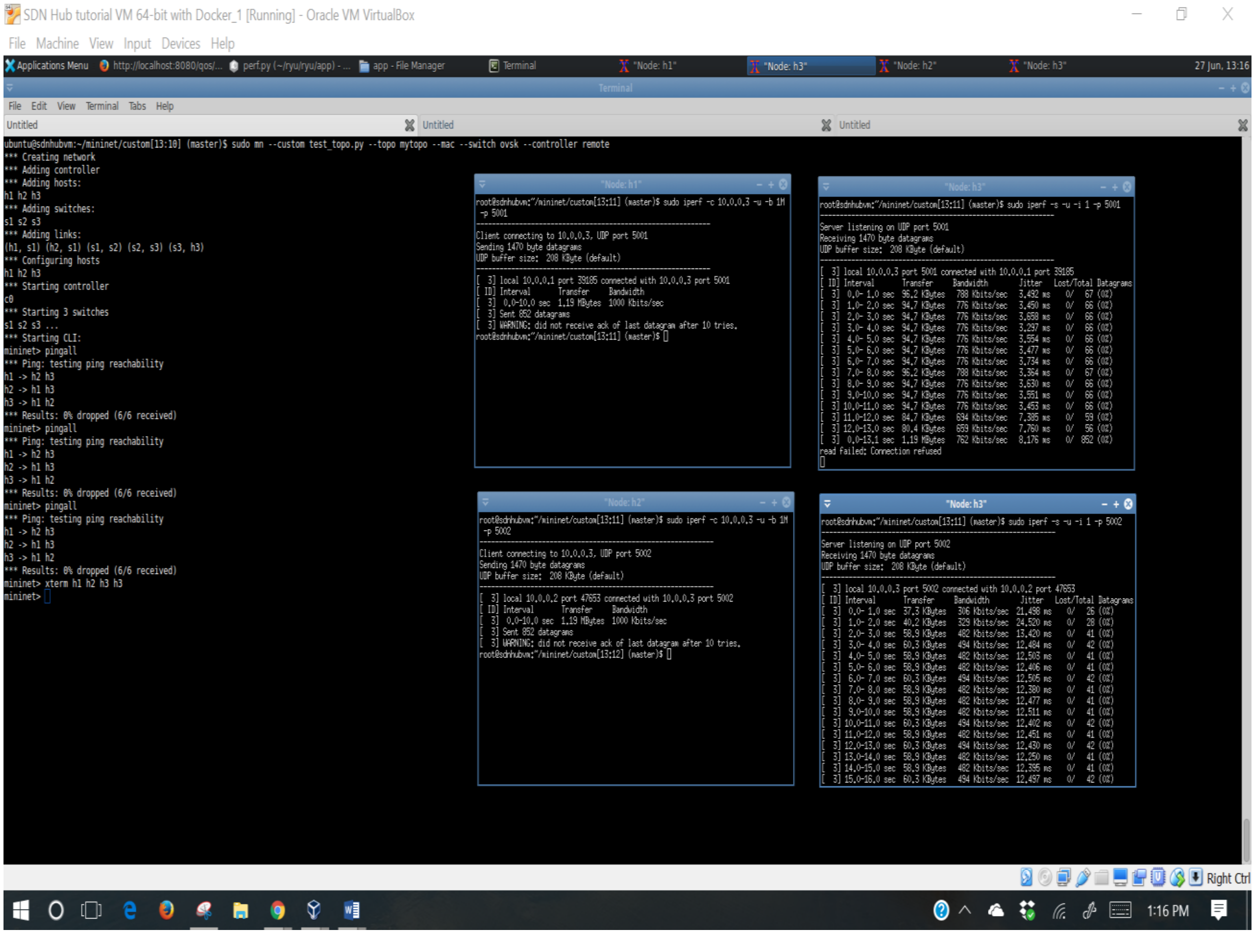
```
▼ 0:
    switch_id:          "0000000000000002"
  ▼ command_result:
      result:           "success"
    ▼ details:
      ▼ 0:
        ▼ config:
            max-rate:   "500000"
      ▼ 1:
        ▼ config:
            max-rate:   "400000"
      ▼ 2:
        ▼ config:
            max-rate:   "800000"
            min-rate:   "600000"
      ▼ 3:
        ▼ config:
            max-rate:   "500000"
            min-rate:   "300000"
```

Packets from 10.0.0.1 (h1) to 10.0.0.3 (h3) are routed through the queue (q0) on s2-eth1 and the packets from 10.0.0.3 (h3) to 10.0.0.1 (h1) are routed through the queue (q2) on s2-eth2 whereas, the packets from 10.0.0.2 (h2) to 10.0.0.3 (h3) are routed through the queue (q1) on s2-eth1 and the packets from 10.0.0.3 (h3) to 10.0.0.2 (h2) are routed through the queue (q3) on s2-eth2.


SDN Hub tutorial VM 64-bit with Docker_1 [Running] - Oracle VM VirtualBox
File Machine View Input Devices Help
Applications Menu
http://localhost:8080/qos/rules/0000000000000002 - Mozilla Firefox
localhost:8080/qos/rules/0000000000000002
JSON
Raw Data
Headers
Save
Copy
Filter JSON
27 Jun, 13:03
Right Ctrl
1:03 PM

```
▼ 0:
    switch_id:              "0000000000000002"
  ▼ command_result:
    ▼ 0:
      ▼ qos:
        ▼ 0:
            priority:       100
            dl_type:        "IPv4"
            nw_dst:         "10.0.0.1"
            qos_id:         1
            nw_src:         "10.0.0.3"
          ▼ actions:
            ▼ 0:
                queue:      "2"
        ▼ 1:
            priority:       100
            dl_type:        "IPv4"
            nw_dst:         "10.0.0.3"
            qos_id:         2
            nw_src:         "10.0.0.1"
          ▼ actions:
            ▼ 0:
                queue:      "0"
        ▼ 2:
            priority:       100
            dl_type:        "IPv4"
            nw_dst:         "10.0.0.2"
            qos_id:         3
            nw_src:         "10.0.0.3"
          ▼ actions:
            ▼ 0:
                queue:      "3"
        ▼ 3:
            priority:       100
            dl_type:        "IPv4"
            nw_dst:         "10.0.0.3"
            qos_id:         4
            nw_src:         "10.0.0.2"
          ▼ actions:
            ▼ 0:
                queue:      "1"
```

We have measured the bandwidth by using iperf, where h3(server) listens on the port 5001 and port 5002. h1(client) and h2(client) sends 1Mbps UDP traffic to the port 5001 on h3 and 1Mbps UDP traffic to the port 5002 on h3.


SDN Hub tutorial VM 64-bit with Docker_1 [Running] - Oracle VM VirtualBox
File Machine View Input Devices Help
Applications Menu
http://localhost:8080/qos/...
perf.py (~/ryu/ryu/app) - ...
app - File Manager
Terminal
"Node: h1"
"Node: h3"
"Node: h2"
"Node: h3"
27 Jun, 13:16
Terminal
File Edit View Terminal Tabs Help
Untitled
Untitled
Untitled
Right Ctrl
1:16 PM

```
ubuntu@sdnhubvm:~/mininet/custom[13:10] (master)$ sudo mn --custom test_topo.py --topo mytopo --mac --switch ovsk --controller remote
*** Creating network
*** Adding controller
*** Adding hosts:
h1 h2 h3
*** Adding switches:
s1 s2 s3
*** Adding links:
(h1, s1) (h2, s1) (s1, s2) (s2, s3) (s3, h3)
*** Configuring hosts
h1 h2 h3
*** Starting controller
c0
*** Starting 3 switches
s1 s2 s3 ...
*** Starting CLI:
mininet> pingall
*** Ping: testing ping reachability
h1 -> h2 h3
h2 -> h1 h3
h3 -> h1 h2
*** Results: 0% dropped (6/6 received)
mininet> pingall
*** Ping: testing ping reachability
h1 -> h2 h3
h2 -> h1 h3
h3 -> h1 h2
*** Results: 0% dropped (6/6 received)
mininet> pingall
*** Ping: testing ping reachability
h1 -> h2 h3
h2 -> h1 h3
h3 -> h1 h2
*** Results: 0% dropped (6/6 received)
mininet> xterm h1 h2 h3 h3
mininet> 
```

"Node: h1"

```
root@sdnhubvm:~/mininet/custom[13:11] (master)$ sudo iperf -c 10.0.0.3 -u -b 1M
 -p 5001
------------------------------------------------------------
Client connecting to 10.0.0.3, UDP port 5001
Sending 1470 byte datagrams
UDP buffer size:  208 KByte (default)
------------------------------------------------------------
[  3] local 10.0.0.1 port 39185 connected with 10.0.0.3 port 5001
[ ID] Interval       Transfer     Bandwidth
[  3]  0.0-10.0 sec  1.19 MBytes  1000 Kbits/sec
[  3] Sent 852 datagrams
[  3] WARNING: did not receive ack of last datagram after 10 tries.
root@sdnhubvm:~/mininet/custom[13:11] (master)$ 
```

"Node: h3"

```
root@sdnhubvm:~/mininet/custom[13:11] (master)$ sudo iperf -s -u -i 1 -p 5001
------------------------------------------------------------
Server listening on UDP port 5001
Receiving 1470 byte datagrams
UDP buffer size:  208 KByte (default)
------------------------------------------------------------
[  3] local 10.0.0.3 port 5001 connected with 10.0.0.1 port 39185
[ ID] Interval       Transfer     Bandwidth        Jitter   Lost/Total Datagrams
[  3]  0.0- 1.0 sec  96.2 KBytes   788 Kbits/sec   3.492 ms    0/   67 (0%)
[  3]  1.0- 2.0 sec  94.7 KBytes   776 Kbits/sec   3.450 ms    0/   66 (0%)
[  3]  2.0- 3.0 sec  94.7 KBytes   776 Kbits/sec   3.658 ms    0/   66 (0%)
[  3]  3.0- 4.0 sec  94.7 KBytes   776 Kbits/sec   3.297 ms    0/   66 (0%)
[  3]  4.0- 5.0 sec  94.7 KBytes   776 Kbits/sec   3.554 ms    0/   66 (0%)
[  3]  5.0- 6.0 sec  94.7 KBytes   776 Kbits/sec   3.477 ms    0/   66 (0%)
[  3]  6.0- 7.0 sec  94.7 KBytes   776 Kbits/sec   3.734 ms    0/   66 (0%)
[  3]  7.0- 8.0 sec  96.2 KBytes   788 Kbits/sec   3.364 ms    0/   67 (0%)
[  3]  8.0- 9.0 sec  94.7 KBytes   776 Kbits/sec   3.630 ms    0/   66 (0%)
[  3]  9.0-10.0 sec  94.7 KBytes   776 Kbits/sec   3.551 ms    0/   66 (0%)
[  3] 10.0-11.0 sec  94.7 KBytes   776 Kbits/sec   3.453 ms    0/   66 (0%)
[  3] 11.0-12.0 sec  84.7 KBytes   694 Kbits/sec   7.385 ms    0/   59 (0%)
[  3] 12.0-13.0 sec  80.4 KBytes   659 Kbits/sec   7.760 ms    0/   56 (0%)
[  3]  0.0-13.1 sec  1.19 MBytes   762 Kbits/sec   8.176 ms    0/  852 (0%)
read failed: Connection refused
```

"Node: h2"

```
root@sdnhubvm:~/mininet/custom[13:11] (master)$ sudo iperf -c 10.0.0.3 -u -b 1M
 -p 5002
------------------------------------------------------------
Client connecting to 10.0.0.3, UDP port 5002
Sending 1470 byte datagrams
UDP buffer size:  208 KByte (default)
------------------------------------------------------------
[  3] local 10.0.0.2 port 47653 connected with 10.0.0.3 port 5002
[ ID] Interval       Transfer     Bandwidth
[  3]  0.0-10.0 sec  1.19 MBytes  1000 Kbits/sec
[  3] Sent 852 datagrams
[  3] WARNING: did not receive ack of last datagram after 10 tries.
root@sdnhubvm:~/mininet/custom[13:12] (master)$ 
```

"Node: h3"

```
root@sdnhubvm:~/mininet/custom[13:11] (master)$ sudo iperf -s -u -i 1 -p 5002
------------------------------------------------------------
Server listening on UDP port 5002
Receiving 1470 byte datagrams
UDP buffer size:  208 KByte (default)
------------------------------------------------------------
[  3] local 10.0.0.3 port 5002 connected with 10.0.0.2 port 47653
[ ID] Interval       Transfer     Bandwidth        Jitter   Lost/Total Datagrams
[  3]  0.0- 1.0 sec  37.3 KBytes   306 Kbits/sec  21.498 ms    0/   26 (0%)
[  3]  1.0- 2.0 sec  40.2 KBytes   329 Kbits/sec  24.520 ms    0/   28 (0%)
[  3]  2.0- 3.0 sec  58.9 KBytes   482 Kbits/sec  13.420 ms    0/   41 (0%)
[  3]  3.0- 4.0 sec  60.3 KBytes   494 Kbits/sec  12.484 ms    0/   42 (0%)
[  3]  4.0- 5.0 sec  58.9 KBytes   482 Kbits/sec  12.503 ms    0/   41 (0%)
[  3]  5.0- 6.0 sec  58.9 KBytes   482 Kbits/sec  12.406 ms    0/   41 (0%)
[  3]  6.0- 7.0 sec  60.3 KBytes   494 Kbits/sec  12.505 ms    0/   42 (0%)
[  3]  7.0- 8.0 sec  58.9 KBytes   482 Kbits/sec  12.380 ms    0/   41 (0%)
[  3]  8.0- 9.0 sec  58.9 KBytes   482 Kbits/sec  12.477 ms    0/   41 (0%)
[  3]  9.0-10.0 sec  58.9 KBytes   482 Kbits/sec  12.511 ms    0/   41 (0%)
[  3] 10.0-11.0 sec  60.3 KBytes   494 Kbits/sec  12.402 ms    0/   42 (0%)
[  3] 11.0-12.0 sec  58.9 KBytes   482 Kbits/sec  12.451 ms    0/   41 (0%)
[  3] 12.0-13.0 sec  60.3 KBytes   494 Kbits/sec  12.430 ms    0/   42 (0%)
[  3] 13.0-14.0 sec  58.9 KBytes   482 Kbits/sec  12.250 ms    0/   41 (0%)
[  3] 14.0-15.0 sec  58.9 KBytes   482 Kbits/sec  12.395 ms    0/   41 (0%)
[  3] 15.0-16.0 sec  60.3 KBytes   494 Kbits/sec  12.497 ms    0/   42 (0%)
```

The above figure shows the implementation results for the Queue management and Bandwidth control using Ryu for the Test scenario 2.

The above result shows that, traffic sent to the port 5001(h1 and h3) is guaranteed with up to 800 Kbps and the traffic to the port 5002 (h2 and h3) is guaranteed with up to 500Kbps bandwidth.

Thus, the above discussions verify that queues are being configured with desired maximum rates and bandwidth is controlled using the Ryu modules perf.py rest_qos.py and rest_conf_switch.py.

## Conclusion:

It can be concluded that, per flow queue configuration and bandwidth control on OVS switches can be performed effectively using a Ryu SDN controller.

The above results show that queue management and bandwidth control using the Ryu controller on OVS switches can be used for restricting particular traffic to a very low bandwidth as well as for allowing particular traffic with a guaranteed maximum available bandwidth.

**References:**

[1] https://osrg.github.io/ryu-book/en/html/rest_qos.html

[2] https://mik.bme.hu/~zfaigl/QoS/doc/README.html

[3] http://ryu.readthedocs.io/en/latest/ryu_app_api.html