

Laboratory work 2

QUALITY OF SERVICE IN SDN NETWORK SCENARIO USING POX CONTROLLER

Goal and objectives: In this laboratory work we will focus on the general principles of quality of services (QoS) in SDN with POX.

Learning objectives:

- study main principles of QoS in SDN;
- study the possibilities of managing SDN with POX controller.

Practical tasks:

- acquire practical skills of network traffic management in SDN;
- acquire practical skills of working with open Vswitch with POX controller.

Exploring tasks:

- exploring the possibilities of Open vSwitch.

Setting up

In preparation for laboratory work it is necessary:

- to clear the goals and mission of the research;
- to study theoretical material contained in this manual, and drill down to [1]-[6];
- to familiarize oneself with the main procedures and specify the exploration program according to defined task.

Recommended software and resources:

- SDNHub

http://yuba.stanford.edu/~srini/tutorial/SDN_tutorial_VM_32bit.ova;

- OracleVM VirtualBOX:

<https://www.oracle.com/technetwork/server-storage/virtualbox/overview/index.html>.

2.1. Synopsis

In this lab we will assess and analyze the Quality of Service of POX Controller in SDN. Furthermore, we outline the potential challenges and open problems that need to be addressed further for better and complete QoS abilities in SDN/OpenFlow networks. Additional information about

available bandwidth measurement in SDN and simulation in SDN network scenario using the POX Controller can be found in [7] and [8] respectively.

2.2. Brief theoretical information

The one of essential features required of IoT networks is to ensure high reliability and Quality of Services (QoS). QoS is typically defined as an ability of a network to provide the required services for selected network traffic. The primary goal of QoS is to provide priority with respect to QoS parameters including but not limited to: bandwidth, delay, jitter, and loss [6].

QoS metrics.

There are the following QoS metrics applicable for the SDN analysis:

1. Network Throughput (NT).

NT is the number of data packets delivered from source to destination per unit of time:

$$NT = \frac{1}{n} \sum_{i=1}^n \frac{b_i}{t_i}, \quad (1)$$

where b_i denotes a total amount of data, t_i is a time taken for destination to get the final packet, n is total number of application traffic.

2. Packet Delivery Ratio (PDR).

PDR is a ratio of the number of packets received (N_{PR}) by the destination to the number of packets send (N_{PS}) by the source:

$$PDR = \frac{N_{PR}}{N_{PC}}. \quad (2)$$

3. Packet Loss (PL).

PL is the measure of number of packets dropped by nodes due to various reasons.

$$PL = N_{PS} - N_{PR}. \quad (22.3)$$

4. Average end-to-end delay (*EED*).

EED is defined as average time taken by data packets to propagate from source to destination across Ad hoc network.

$$EED = \sum (\text{arrived time} - \text{sent time}). \quad (4)$$

Performance tests.

1. Network connectivity test

Pingall stamen checks the connectivity among the created network. Connectivity among the hosts ensures the data transfer is possible among them. Wget feature of Linux has been used for sending files among the reachable hosts. Fig. 15 depicts connectivity among the hosts depending on the congestion state at the particular time. Frame 1 shows that h1 and h3 has connectivity to all other hosts, whereas from h2 there is no connectivity to h1 and h4 has no connectivity to h1. Frame 2 shows that h2 has no reachability to h3 and h3 has no reachability to h4.

```
*** Ping: testing ping reachability
h1 -> h2 h3 h4
h2 -> X h3 h4
h3 -> h1 h2 h4
h4 -> X h2 h3
*** Ping: testing ping reachability
```

Fig. 15 – Pingall reachability test

2. SDN available bandwidth measurement

Bandwidth utilization in a network serves as a key method for measuring QoS of network. Available bandwidth is an important component for both service provider and application perspective [7].

Fig. 16 shows the two bandwidth frames taken at different amount of time. It measures the bandwidth between h1 and h4 at different available capacity links.

```
Testing bandwidth between h1 and h4
*** Iperf: testing TCP bandwidth between h1 and h4
*** Results: ['3.03 Mbits/sec' , '3.15 Mbits/sec']
testing bandwidth between h1 and h4
```

```
*** Iperf: testing TCP bandwidth between h1 and h4
*** Results: ['3.52 Mbits/sec' , '3.66 Mbits/sec']
```

Fig. 16 – Available bandwidth measurement

3. Packet loss and delay measurement

Packet loss ratio and packet delay will benefit many users and operators of network applications. Fig. 17 shows the example of packet loss and delay in the SDN network.

```
*** Adding links:
(10.00Mbit 5ms delay 2% loss) (10.00Mbit 5ms delay 2% loss) (h1, s1)
(10.00Mbit 5ms delay 2% loss) (10.00Mbit 5ms delay 2% loss) (h2, s1)
(10.00Mbit 5ms delay 2% loss) (10.00Mbit 5ms delay 2% loss) (h3, s1)
(10.00Mbit 5ms delay 2% loss) (10.00Mbit 5ms delay 2% loss) (h4, s1)
Packet loss
*** Results: 16% dropped (10/12 received)
```

Fig. 17 – Packet loss and delay in the SDN network

2.3. Execution order and discovery questions

The simulation scenario consists of two OpenFlow switches (S1 and S2) connected to the three hosts (h1, h2, h3) and to the POX controller as depicted in Fig. 18.

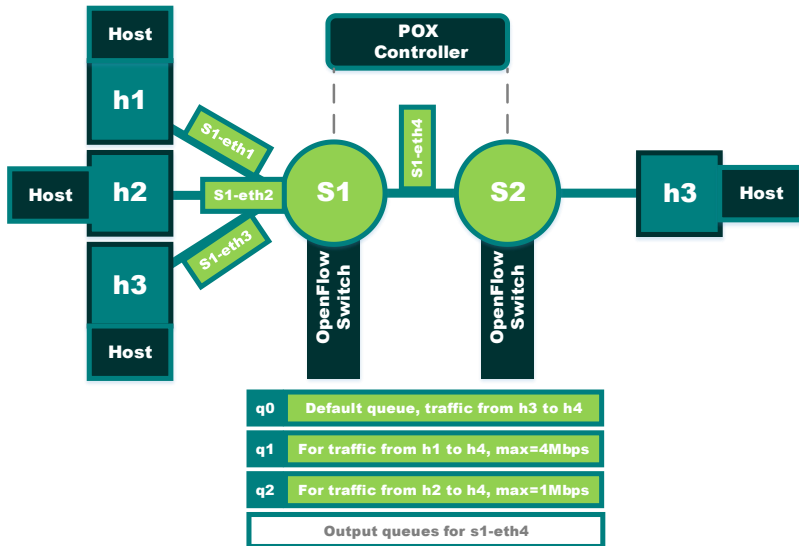


Fig. 18 – Example of network under the test

First, you need to install Linux to Oracle VM Virtual Box and deploy SDNHub ova image as you did it in previous laboratory work.

Step 1. Configure openVswitch with POX controller.

To configure openVswitch in PC1 eth0.10 interface you can use the following commands [6]:

1. Attach PC1 eth0.10 interface (IP 192.168.10.100) to the bridge connection between openVswitch in PC1 and controller.

```
> sudo ovs-vsctl add-br br0
> sudo ovs-vsctl add-port br0 eth0.10
> sudo ifconfig br0 192.168.10.100 netmask 255.255.255.0
```

2. Attach OpenVswitch to the Controller which is in 192.168.100.30.

```
> ovs-vsctl set-controller br0 tcp:192.168.100.30:6633
```

3. To remove openVswitch bridge, connection use the following command.

```
> sudo ovs-vsctl del-br br-0
> sudo ovs-vsctl del-port br-0 eth0.10
```

4. To remove the Controller type, do the following.

```
> sudo ovs-vsctl del-controller br-0
```

5. To launch multiple controllers and *Mininet* on the same VM, run *Mininet* Script given below.

```
> c1 = net.addController('c1',
controller=RemoteController, ip="127.0.0.2",
port=6633)
> c2 = net.addController('c2',
controller=RemoteController, ip="127.0.0.1",
port=6634)
```

6. Start POX controllers.

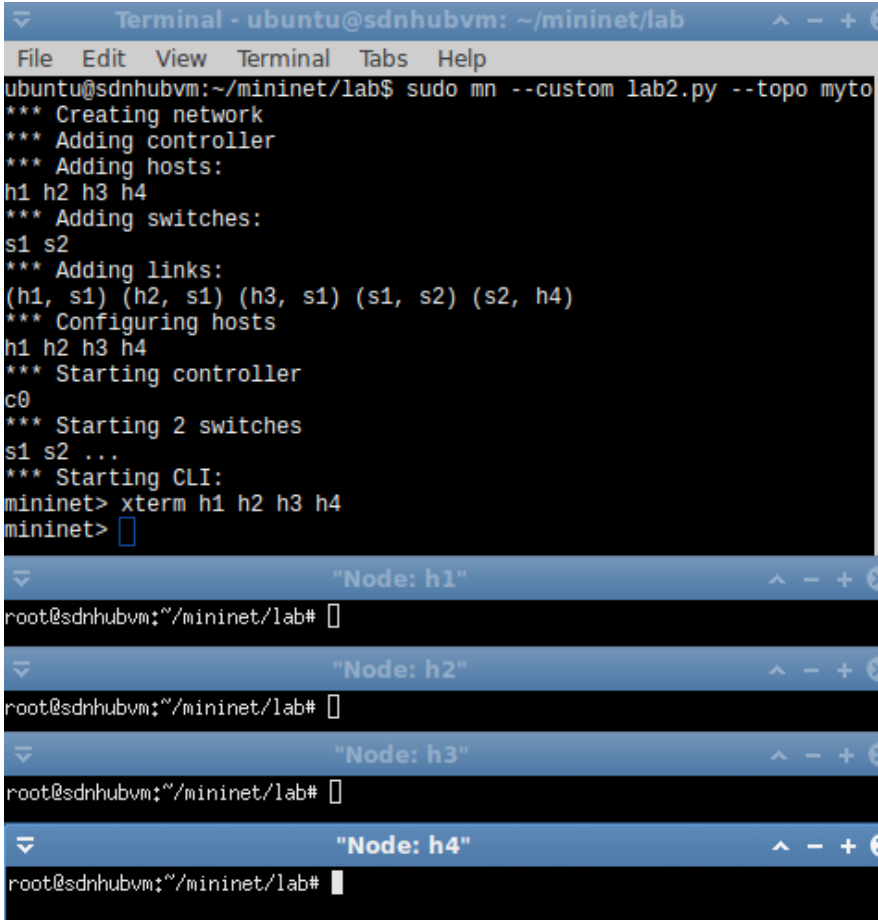
```
> ./pox.py --port=6633 MyScript.py
> ./pox.py --port=6634 MyScript.py
```

Step 2. Build the network from the Fig. 15.

```
> sudo mn --custom lab2.py --topo mytopo -mac
```

Step 3. Call command prompt h1 and h2.

```
> xterm h1 h2 h3 h4
```



```
Terminal - ubuntu@sdnhubvm: ~/mininet/lab
File Edit View Terminal Tabs Help
ubuntu@sdnhubvm:~/mininet/lab$ sudo mn --custom lab2.py --topo myto
*** Creating network
*** Adding controller
*** Adding hosts:
h1 h2 h3 h4
*** Adding switches:
s1 s2
*** Adding links:
(h1, s1) (h2, s1) (h3, s1) (s1, s2) (s2, h4)
*** Configuring hosts
h1 h2 h3 h4
*** Starting controller
c0
*** Starting 2 switches
s1 s2 ...
*** Starting CLI:
mininet> xterm h1 h2 h3 h4
mininet> 
```

```
"Node: h1"
root@sdnhubvm:~/mininet/lab# 
```

```
"Node: h2"
root@sdnhubvm:~/mininet/lab# 
```

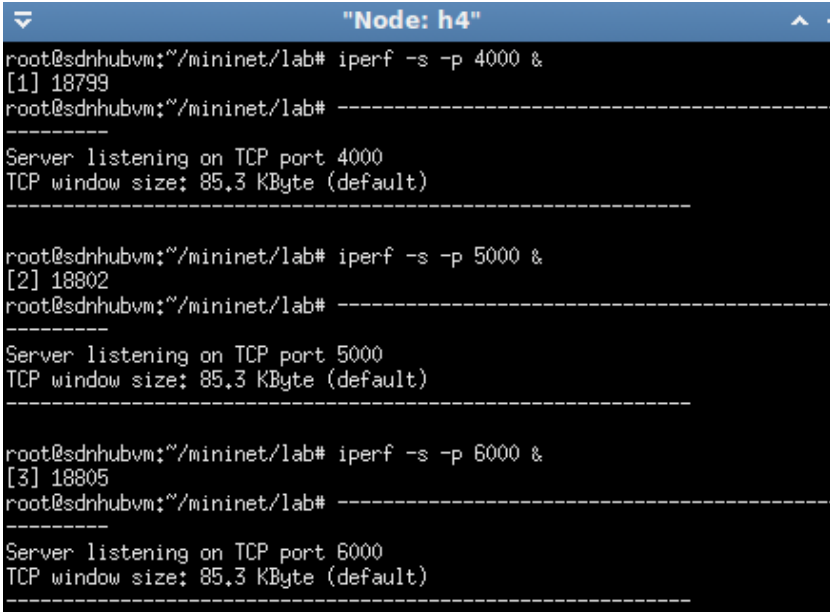
```
"Node: h3"
root@sdnhubvm:~/mininet/lab# 
```

```
"Node: h4"
root@sdnhubvm:~/mininet/lab# 
```

Fig. 19 – Creating the network

Step 4. In h4 windows, start Iperf servers at port 4000, 5000, 6000 respectively.

```
> iperf -s -p 4000&
iperf -s -p 5000&
iperf -s -p 6000
```



```

Node: h4
root@sdnhubvm:~/mininet/lab# iperf -s -p 4000 &
[1] 18799
root@sdnhubvm:~/mininet/lab# -----
Server listening on TCP port 4000
TCP window size: 85.3 KByte (default)
-----

root@sdnhubvm:~/mininet/lab# iperf -s -p 5000 &
[2] 18802
root@sdnhubvm:~/mininet/lab# -----
Server listening on TCP port 5000
TCP window size: 85.3 KByte (default)
-----

root@sdnhubvm:~/mininet/lab# iperf -s -p 6000 &
[3] 18805
root@sdnhubvm:~/mininet/lab# -----
Server listening on TCP port 6000
TCP window size: 85.3 KByte (default)
-----

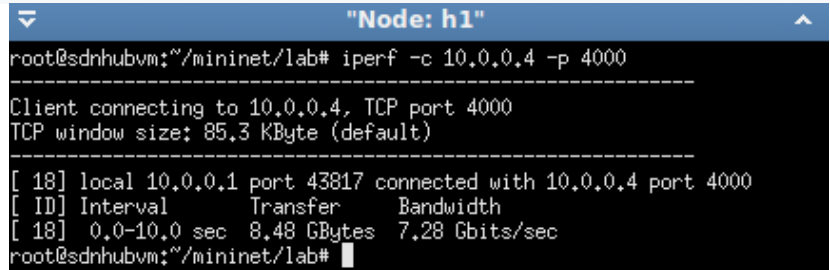
```

Fig. 20 – Server listening

Step 5. Conduct different tests to check the performance of SDN network. Besides the performance tests, you can use the following.

1. Node h1. Test the bandwidth between h1 and h4 (no other background traffic)

```
> iperf -c 10.0.0.4 -p 4000
```



```

Node: h1
root@sdnhubvm:~/mininet/lab# iperf -c 10.0.0.4 -p 4000
-----
Client connecting to 10.0.0.4, TCP port 4000
TCP window size: 85.3 KByte (default)
-----

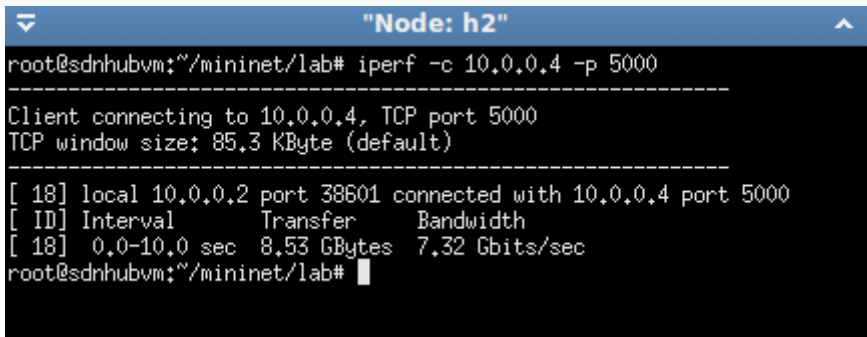
[ 18] local 10.0.0.1 port 43817 connected with 10.0.0.4 port 4000
[ ID] Interval      Transfer    Bandwidth
[ 18] 0.0-10.0 sec  8.48 GBytes  7.28 Gbits/sec
root@sdnhubvm:~/mininet/lab# █

```

Fig. 21 – Test the bandwidth between h1 and h4

2. Node h2. Test the bandwidth between h2 and h4 (no other background traffic).

```
> iperf -c 10.0.0.4 -p 5000
```



```

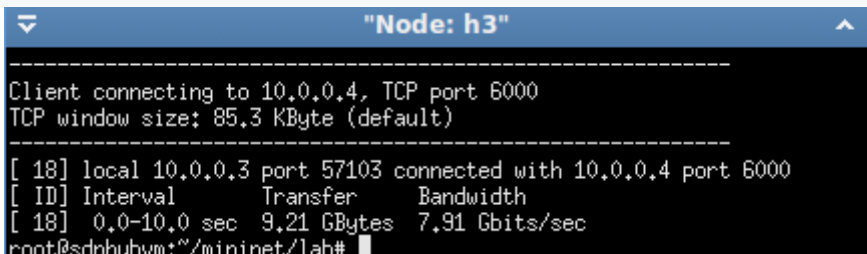
"Node: h2"
root@sdnhubvm:~/mininet/lab# iperf -c 10.0.0.4 -p 5000
-----
Client connecting to 10.0.0.4, TCP port 5000
TCP window size: 85.3 KByte (default)
-----
[ 18] local 10.0.0.2 port 38601 connected with 10.0.0.4 port 5000
[ ID] Interval      Transfer    Bandwidth
[ 18] 0.0-10.0 sec  8.53 GBytes 7.32 Gbits/sec
root@sdnhubvm:~/mininet/lab#

```

Fig. 22 – Test the bandwidth between h2 t and h4

3. Test the bandwidth between h3 and h4 (no other background traffic).

```
> iperf -c 10.0.0.4 -p 6000
```



```

"Node: h3"
Client connecting to 10.0.0.4, TCP port 6000
TCP window size: 85.3 KByte (default)
-----
[ 18] local 10.0.0.3 port 57103 connected with 10.0.0.4 port 6000
[ ID] Interval      Transfer    Bandwidth
[ 18] 0.0-10.0 sec  9.21 GBytes 7.91 Gbits/sec
root@sdnhubvm:~/mininet/lab#

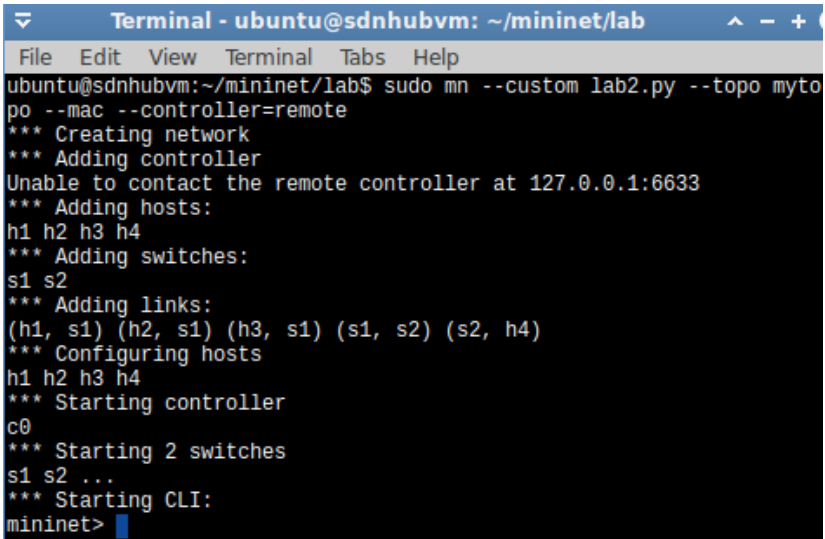
```

Fig. 23 – Test the bandwidth between h3 and h4

The measured bandwidth will be around 7.91 Gbits/sec. They depend from the emulation environment, such as CPU and working load.

Step 6. Restart Mininet.

```
> sudo mn --custom lab2.py --topo mytopo -mac --
controller remote
```



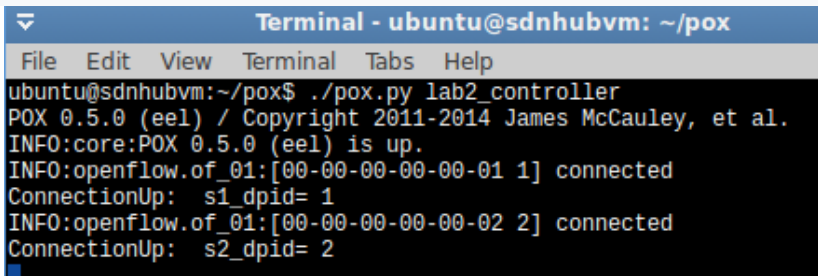
```
Terminal - ubuntu@sdnhubvm: ~/mininet/lab
File Edit View Terminal Tabs Help
ubuntu@sdnhubvm:~/mininet/lab$ sudo mn --custom lab2.py --topo myto
po --mac --controller=remote
*** Creating network
*** Adding controller
Unable to contact the remote controller at 127.0.0.1:6633
*** Adding hosts:
h1 h2 h3 h4
*** Adding switches:
s1 s2
*** Adding links:
(h1, s1) (h2, s1) (h3, s1) (s1, s2) (s2, h4)
*** Configuring hosts
h1 h2 h3 h4
*** Starting controller
c0
*** Starting 2 switches
s1 s2 ...
*** Starting CLI:
mininet>
```

Fig. 24 – Restart Mininet

Step 7. Put lab2_controller.py to home/ubuntu/pox/ext/.

Step 8. Start POX controller.

```
> cd pox
> ./pox.py lab2_controller
```



```
Terminal - ubuntu@sdnhubvm: ~/pox
File Edit View Terminal Tabs Help
ubuntu@sdnhubvm:~/pox$ ./pox.py lab2_controller
POX 0.5.0 (eel) / Copyright 2011-2014 James McCauley, et al.
INFO:core:POX 0.5.0 (eel) is up.
INFO:openflow.of_01:[00-00-00-00-00-01 1] connected
ConnectionUp: s1_dpid= 1
INFO:openflow.of_01:[00-00-00-00-00-02 2] connected
ConnectionUp: s2_dpid= 2
```

Fig. 25 – Start POX controller

Step 9. Set up Open vSwitch for queues. Create a linux-htb QoS record that points to a few queues and use it on eth4.

Step 10. Create three queues for s1-eth4, i.e. q0, q1, and q2 and to set the rate for each queue using ovs-vsctl.

To do this, enter the following command (Fig. 26):

```
> sudo ovs-vsctl -- set Port s1-eth4 qos=@newqos -- --id=@newqos create QoS type=linux-htb other-config:max-rate=100000000 queues=0=@q0,1=@q1,2=@q2 -- --id=@q0 create Queue other-config:min-rate=1000000000 other-config:max-rate=1000000000 -- --id=@q1 create Queue other-config:min-rate=4000000 other-config:max-rate=4000000 -- --id=@q2 create Queue other-config:min-rate=100000 other-config:max-rate=1000000
```

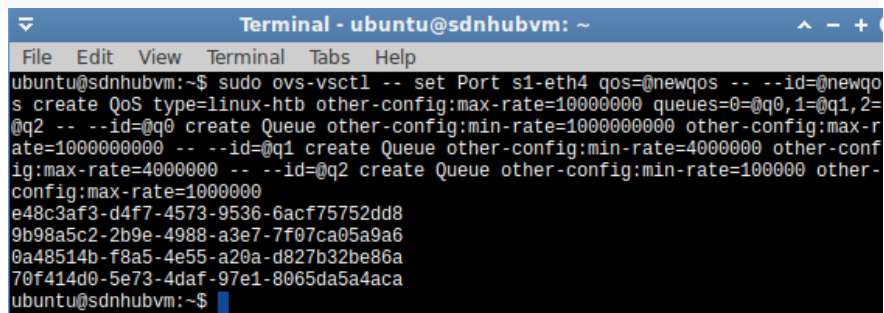


Fig. 26 – Creating queries for s1-eth4

More detailed information about Open vSwitch is given here: <http://www.openvswitch.org/support/dist-docs/ovs-vsctl.8.txt>.

Node h4.

```
> iperf -s -p 4000 &
iperf -s -p 5000 &
iperf -s -p 6000
```

Node h1.

```
> iperf -c 10.0.0.4 -p 4000
```

Node h2.

```
> iperf -c 10.0.0.4 -p 5000
```

Node h3.

```
> iperf -c 10.0.0.4 -p 6000
```

As you can see (Fig. 27), the bandwidth between the hosts have been changed. This is a result of performed operations.

Node: h1	Node: h4
<pre>root@sdrhubvm:/mininet/lab# iperf -c 10.0.0.4 -p 4000 Client connecting to 10.0.0.4, TCP port 4000 TCP window size: 85.3 KByte (default) [18] local 10.0.0.1 port 48270 connected with 10.0.0.4 port 4000 [ID] Interval Transfer Bandwidth [18] 0.0-10.2 sec 5.62 MBytes 4.62 Mbits/sec root@sdrhubvm:/mininet/lab#</pre>	<pre>root@sdrhubvm:/mininet/lab# iperf -s -p 5000 & [2] 20096 root@sdrhubvm:/mininet/lab# Server listening on TCP port 5000 TCP window size: 85.3 KByte (default) root@sdrhubvm:/mininet/lab# iperf -s -p 6000 Server listening on TCP port 6000 TCP window size: 85.3 KByte (default) [19] local 10.0.0.4 port 4000 connected with 10.0.0.1 port 48270 [ID] Interval Transfer Bandwidth [19] 0.0-12.5 sec 5.62 MBytes 3.77 Mbits/sec [19] local 10.0.0.4 port 5000 connected with 10.0.0.2 port 43055 [ID] Interval Transfer Bandwidth [19] 0.0-17.5 sec 2.00 MBytes 957 Kbits/sec [19] local 10.0.0.4 port 6000 connected with 10.0.0.3 port 33324 [ID] Interval Transfer Bandwidth [19] 0.0-11.7 sec 12.1 MBytes 8.69 Mbits/sec </pre>
<pre>root@sdrhubvm:/mininet/lab# iperf -c 10.0.0.4 -p 5000 Client connecting to 10.0.0.4, TCP port 5000 TCP window size: 85.3 KByte (default) [18] local 10.0.0.2 port 43055 connected with 10.0.0.4 port 5000 [ID] Interval Transfer Bandwidth [18] 0.0-11.0 sec 2.00 MBytes 1.53 Mbits/sec root@sdrhubvm:/mininet/lab#</pre>	<pre>root@sdrhubvm:/mininet/lab# iperf -c 10.0.0.4 -p 6000 Client connecting to 10.0.0.4, TCP port 6000 TCP window size: 85.3 KByte (default) [18] local 10.0.0.3 port 33324 connected with 10.0.0.4 port 6000 [ID] Interval Transfer Bandwidth [18] 0.0-10.3 sec 12.1 MBytes 9.32 Mbits/sec root@sdrhubvm:/mininet/lab#</pre>

Fig. 27 – New values of the bandwidth between the hosts

After executing all available performance tests, analyze the possible changes and their causes.

2.4. Requirements for the content of the report

Report should contain 5 sections: Introduction (I), Methods (M), Results (R), and Discussion (D)

- (I): background / theory, purpose and discovery questions
- (M): complete description of the software, and procedures which was followed in the experiment, experiment overview, procedures

- (R): narrate (like a story), code for your assignment, Fig. of attack graph with marked the attacker path;
- (D): answers on discovery questions, explanation of results, conclusion / summary.

2.5. Control questions:

1. What network performance metrics in SDN do you know?
2. How to set up Open vSwitch for queues?
3. What are the key tests for measuring QoS of network with POX?
4. How to check the performance of SDN network?
5. What parameters does measured bandwidth depend on?
6. How to perform network connectivity test?
7. How to obtain measure of connectivity among the hosts depending on the congestion state at the particular time?
8. How does network architecture affect bandwidth between nodes?
9. What are the main causes of packet loss and delay measurement in SDN?
10. How to overcome issues with packet loss and delay measurement in SDN?

2.6. Recommended literature:

1. "Basic Configuration – Open vSwitch 2.12.90 documentation", *Docs.openvswitch.org*, 2019. [Online]. Available: <http://docs.openvswitch.org/en/latest/faq/configuration/>. [Accessed: 13 Jun. 2019].
2. QoS on OpenFlow 1.0 with OVS 1.4.3 and POX inside Mininet, 2018. [Online]. Available: http://users.ecs.soton.ac.uk/drn/ofertie/openflow_qos_mininet.pdf [Accessed: 12- Sep. 2018].
3. Open vSwitch 2018. [Online]. Available: <http://www.openvswitch.org/support/dist-docs/ovs-vsctl.8.txt> [Accessed: 12 Sep. 2018].
4. "POX Controller Tutorial | SDN Hub", *Sdnhub.org*, 2019. [Online]. Available: <http://sdnhub.org/tutorials/pox/>. [Accessed: 23 Jun. 2019].
5. "Configure openVswitch with POX controller", *Windysdn.blogspot.com*, 2019. [Online]. Available: <http://windysdn.blogspot.com/2013/10/configure-openvswitch-with-pox.html>. [Accessed: 03 Jan. 2019].

6. S. Midha, K. Tripathi, "Assessing the Quality of Service of POX Controller in SDN," *International Journal of Computational Engineering Research (IJCER)*, 2018, vol. 8, no. 8, pp. 21-25.
7. P. Megyesi, A. Botta, G. Aceto, A. Pescapè, S. Molnár, "Available Bandwidth measurement in SDN", *ACM* 978-1-4503-3739-7/16/04
8. L. R. Prete, A. Shinoda, C. Schweitzer and R. de Oliveira, "Simulation in an SDN network scenario using the POX Controller", *2014 IEEE Colombian Conference on Communications and Computing (COLCOM)*, 2014. doi: 10.1109/colcomcon.2014.6860403.