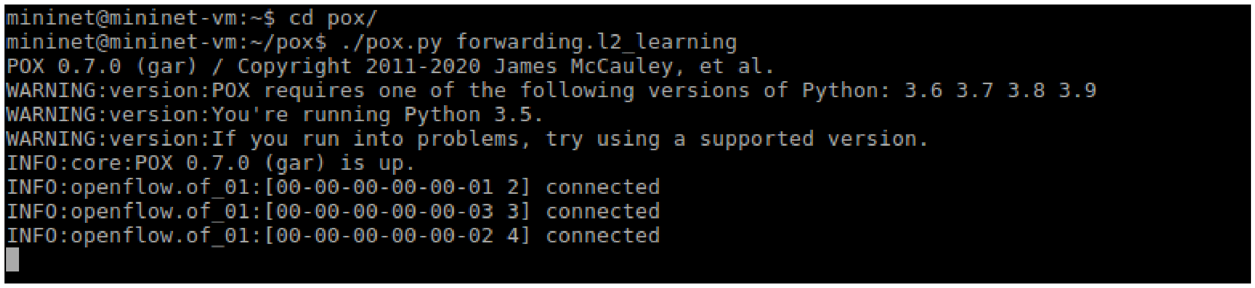
Assignment 4

Shubham Sharma, 2021099

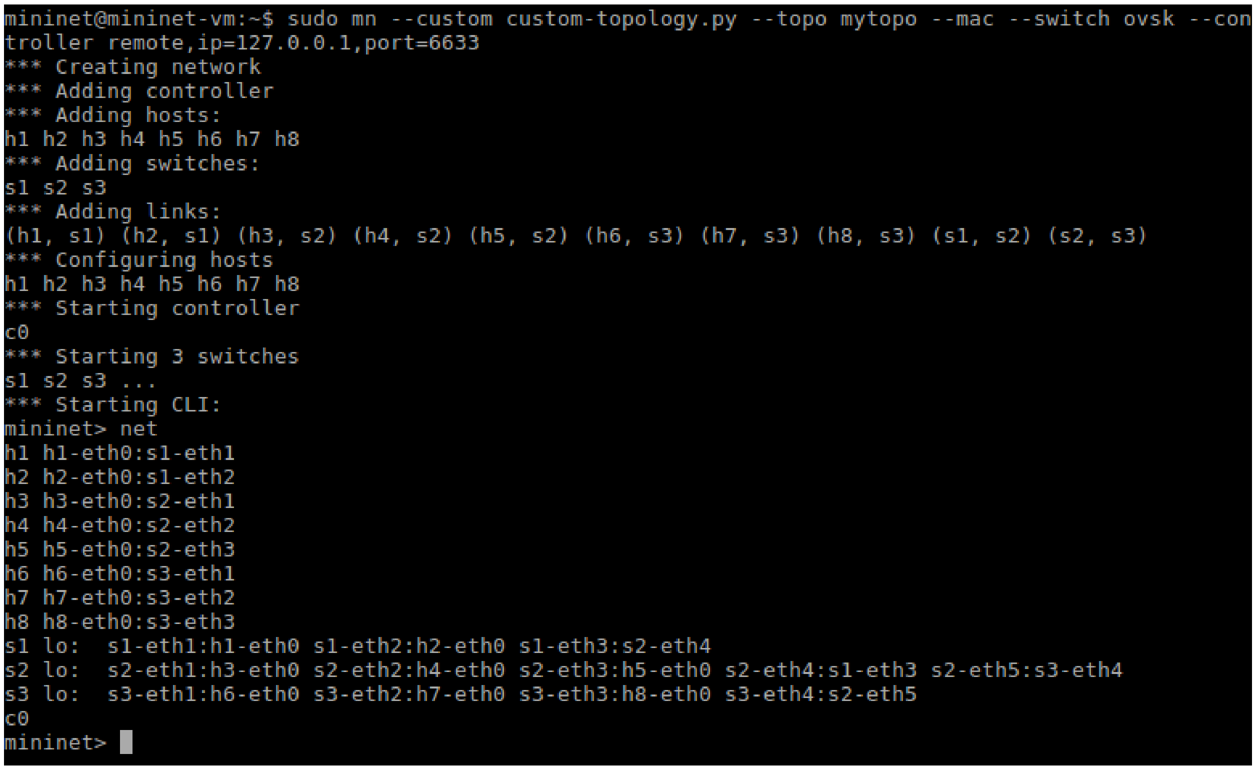
1) a), 1) b)

POX Controller



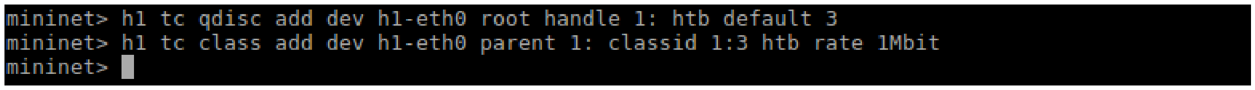
```
mininet@mininet-vm:~$ cd pox/
mininet@mininet-vm:~/pox$ ./pox.py forwarding.l2_learning
POX 0.7.0 (gar) / Copyright 2011-2020 James McCauley, et al.
WARNING:version:POX requires one of the following versions of Python: 3.6 3.7 3.8 3.9
WARNING:version:You're running Python 3.5.
WARNING:version:If you run into problems, try using a supported version.
INFO:core:POX 0.7.0 (gar) is up.
INFO:openflow.of_01:[00-00-00-00-00-01 2] connected
INFO:openflow.of_01:[00-00-00-00-00-03 3] connected
INFO:openflow.of_01:[00-00-00-00-00-02 4] connected
```

Topology



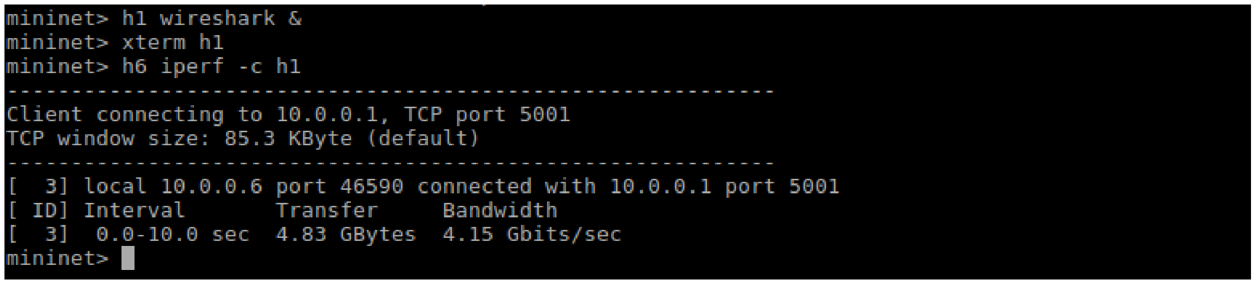
```
mininet@mininet-vm:~$ sudo mn --custom custom-topology.py --topo mytopo --mac --switch ovsk --con
troller remote,ip=127.0.0.1,port=6633
*** Creating network
*** Adding controller
*** Adding hosts:
h1 h2 h3 h4 h5 h6 h7 h8
*** Adding switches:
s1 s2 s3
*** Adding links:
(h1, s1) (h2, s1) (h3, s2) (h4, s2) (h5, s2) (h6, s3) (h7, s3) (h8, s3) (s1, s2) (s2, s3)
*** Configuring hosts
h1 h2 h3 h4 h5 h6 h7 h8
*** Starting controller
c0
*** Starting 3 switches
s1 s2 s3 ...
*** Starting CLI:
mininet> net
h1 h1-eth0:s1-eth1
h2 h2-eth0:s1-eth2
h3 h3-eth0:s2-eth1
h4 h4-eth0:s2-eth2
h5 h5-eth0:s2-eth3
h6 h6-eth0:s3-eth1
h7 h7-eth0:s3-eth2
h8 h8-eth0:s3-eth3
s1 lo:  s1-eth1:h1-eth0 s1-eth2:h2-eth0 s1-eth3:s2-eth4
s2 lo:  s2-eth1:h3-eth0 s2-eth2:h4-eth0 s2-eth3:h5-eth0 s2-eth4:s1-eth3 s2-eth5:s3-eth4
s3 lo:  s3-eth1:h6-eth0 s3-eth2:h7-eth0 s3-eth3:h8-eth0 s3-eth4:s2-eth5
c0
mininet> 
```

2) a) Bottleneck using the “TC” command

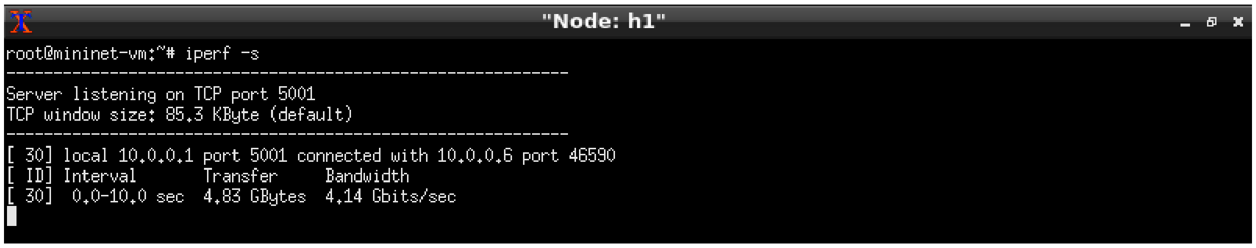


```
mininet> h1 tc qdisc add dev h1-eth0 root handle 1: htb default 3
mininet> h1 tc class add dev h1-eth0 parent 1: classid 1:3 htb rate 1Mbit
mininet> 
```

2) b), 2) c) TCP traffic between the hosts “h1” and “h6” using Iperf and packet capture on host “h1” using Wireshark.

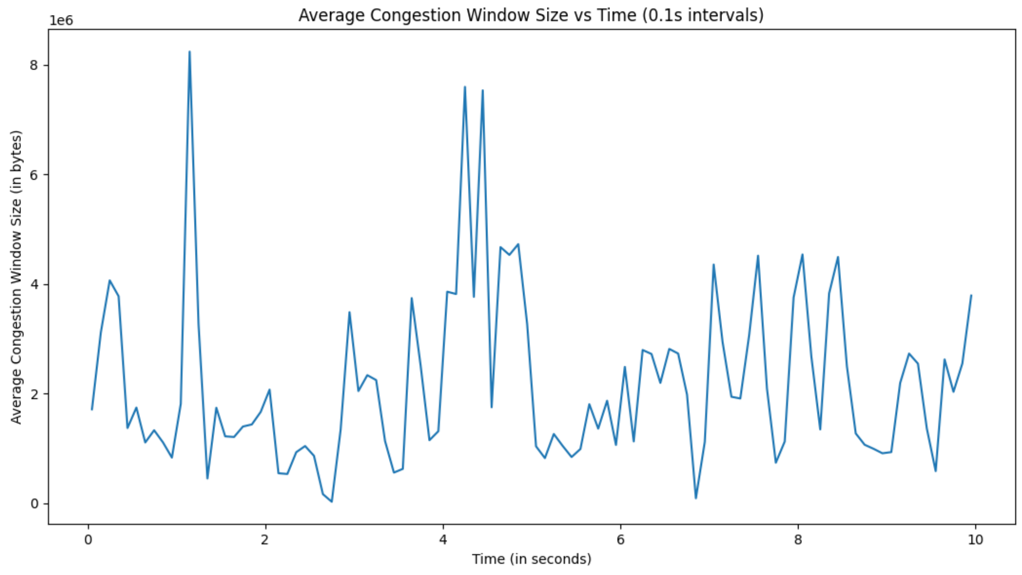


```
mininet> h1 wireshark &
mininet> xterm h1
mininet> h6 iperf -c h1
------------------------------------------------------------
Client connecting to 10.0.0.1, TCP port 5001
TCP window size: 85.3 KByte (default)
------------------------------------------------------------
[  3] local 10.0.0.6 port 46590 connected with 10.0.0.1 port 5001
[ ID] Interval       Transfer     Bandwidth
[  3]  0.0-10.0 sec  4.83 GBytes  4.15 Gbits/sec
mininet> 
```



```
"Node: h1"
root@mininet-vm:~# iperf -s
------------------------------------------------------------
Server listening on TCP port 5001
TCP window size: 85.3 KByte (default)
------------------------------------------------------------
[ 30] local 10.0.0.1 port 5001 connected with 10.0.0.6 port 46590
[ ID] Interval       Transfer     Bandwidth
[ 30]  0.0-10.0 sec  4.83 GBytes  4.14 Gbits/sec
```

2) d) Congestion Window vs Time Plot



In the plot, I observed a **Sawtooth behaviour**. As it is clearly visible, it starts from the slow start phase, and then, for every acknowledged packet, it doubles its congestion window. If, at any point, any packet loss or congestion occurs, it backs off to alleviate congestion, which results in a downward slope. In the case of duplicate Acks, instead of a slow start phase, it uses a Fast recovery mechanism and retransmits the next packets in sequence without waiting for a timeout. All these mechanisms together generate a Sawtooth pattern.