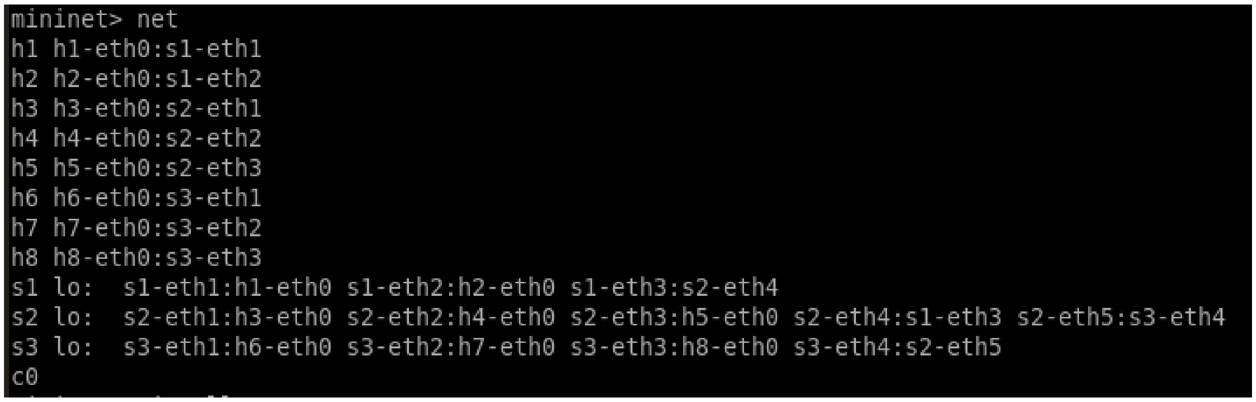
CN Assignment 4 Report

Srimant Mohanty
2021207
srimant21207@iiitd.ac.in

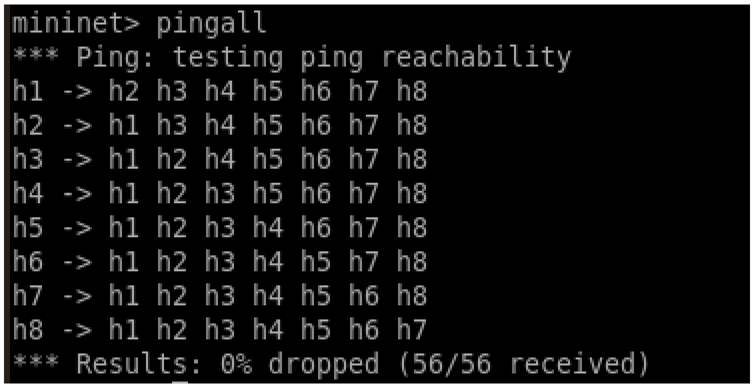
Q1.

a) Custom Topology created using Python Code.
b) Network Configuration:



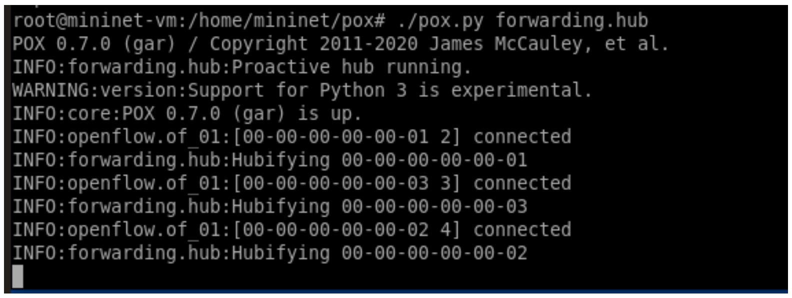
```
mininet> net
h1 h1-eth0:s1-eth1
h2 h2-eth0:s1-eth2
h3 h3-eth0:s2-eth1
h4 h4-eth0:s2-eth2
h5 h5-eth0:s2-eth3
h6 h6-eth0:s3-eth1
h7 h7-eth0:s3-eth2
h8 h8-eth0:s3-eth3
s1 lo:  s1-eth1:h1-eth0 s1-eth2:h2-eth0 s1-eth3:s2-eth4
s2 lo:  s2-eth1:h3-eth0 s2-eth2:h4-eth0 s2-eth3:h5-eth0 s2-eth4:s1-eth3 s2-eth5:s3-eth4
s3 lo:  s3-eth1:h6-eth0 s3-eth2:h7-eth0 s3-eth3:h8-eth0 s3-eth4:s2-eth5
c0
```

All hosts can be pinged



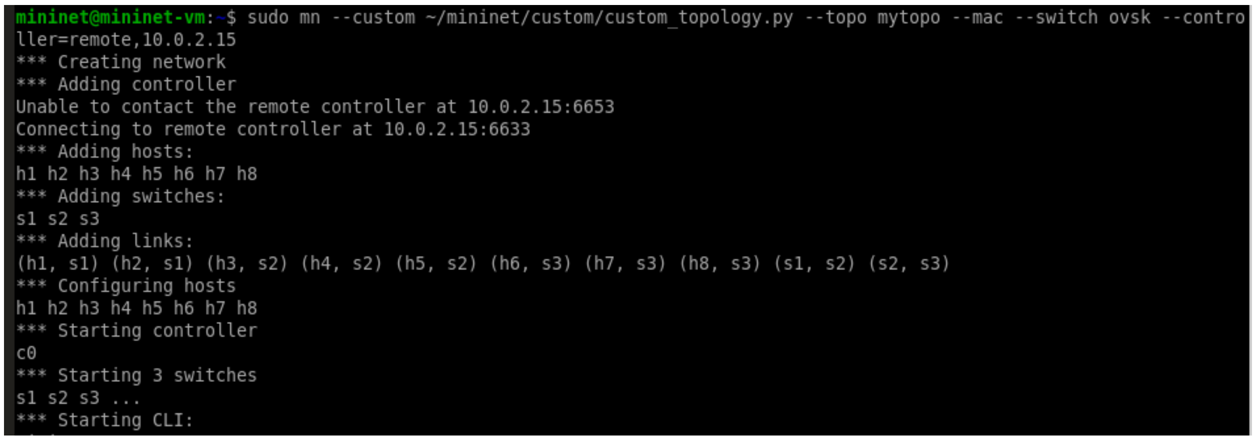
```
mininet> pingall
*** Ping: testing ping reachability
h1 -> h2 h3 h4 h5 h6 h7 h8
h2 -> h1 h3 h4 h5 h6 h7 h8
h3 -> h1 h2 h4 h5 h6 h7 h8
h4 -> h1 h2 h3 h5 h6 h7 h8
h5 -> h1 h2 h3 h4 h6 h7 h8
h6 -> h1 h2 h3 h4 h5 h7 h8
h7 -> h1 h2 h3 h4 h5 h6 h8
h8 -> h1 h2 h3 h4 h5 h6 h7
*** Results: 0% dropped (56/56 received)
```

POX Remote Controller added for the topology



```
root@mininet-vm:/home/mininet/pox# ./pox.py forwarding.hub
POX 0.7.0 (gar) / Copyright 2011-2020 James McCauley, et al.
INFO:forwarding.hub:Proactive hub running.
WARNING:version:Support for Python 3 is experimental.
INFO:core:POX 0.7.0 (gar) is up.
INFO:openflow.of_01:[00-00-00-00-00-01 2] connected
INFO:forwarding.hub:Hubifying 00-00-00-00-00-01
INFO:openflow.of_01:[00-00-00-00-00-03 3] connected
INFO:forwarding.hub:Hubifying 00-00-00-00-00-03
INFO:openflow.of_01:[00-00-00-00-00-02 4] connected
INFO:forwarding.hub:Hubifying 00-00-00-00-00-02
```

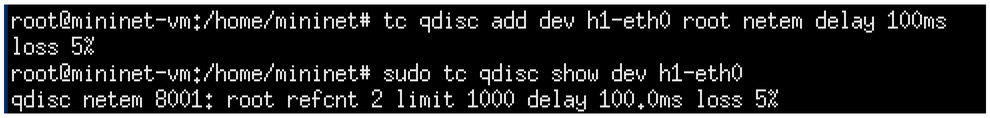
Topology Created, Open VSwitches added & controller all connected.



```
mininet@mininet-vm:~$ sudo mn --custom ~/mininet/custom/custom_topology.py --topo mytopo --mac --switch ovsk --contro
ller=remote,10.0.2.15
*** Creating network
*** Adding controller
Unable to contact the remote controller at 10.0.2.15:6653
Connecting to remote controller at 10.0.2.15:6633
*** 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:
```

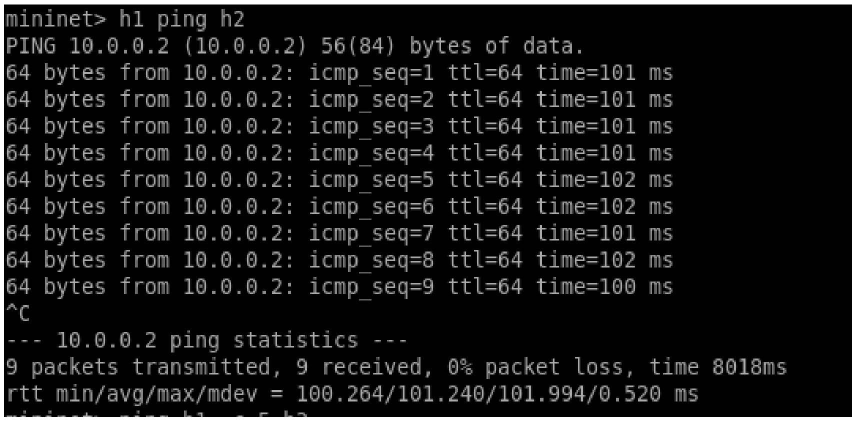
Q2. a)
Creating a bottleneck at host h1 using ‘tc’ commands which is used to manipulate the traffic control settings.

This command creates a bottleneck at the host h1-eth0 interface by introducing a delay of 100ms and a loss of 5%.
The next command basically shows the current configuration for the host.



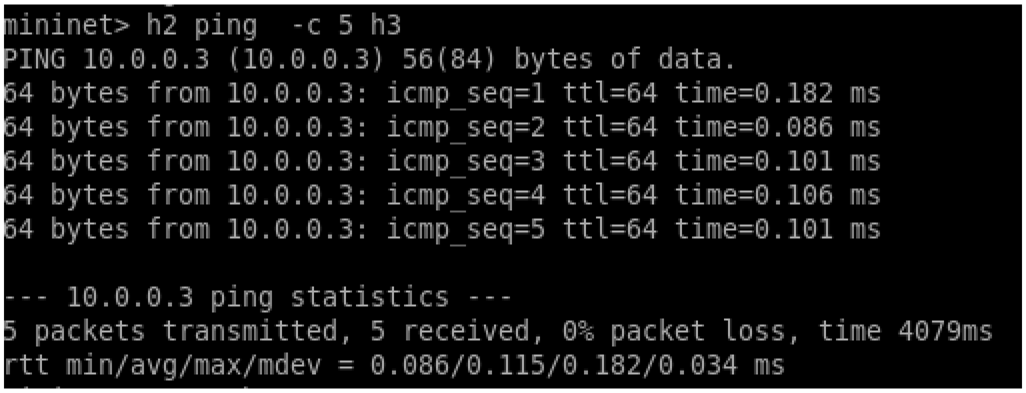
```
root@mininet-vm:/home/mininet# tc qdisc add dev h1-eth0 root netem delay 100ms
loss 5%
root@mininet-vm:/home/mininet# sudo tc qdisc show dev h1-eth0
qdisc netem 8001: root refcnt 2 limit 1000 delay 100.0ms loss 5%
```

The above case of creating a bottleneck can also be verified by pinging the hosts through the mininet terminal.



```
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_seq=1 ttl=64 time=101 ms
64 bytes from 10.0.0.2: icmp_seq=2 ttl=64 time=101 ms
64 bytes from 10.0.0.2: icmp_seq=3 ttl=64 time=101 ms
64 bytes from 10.0.0.2: icmp_seq=4 ttl=64 time=101 ms
64 bytes from 10.0.0.2: icmp_seq=5 ttl=64 time=102 ms
64 bytes from 10.0.0.2: icmp_seq=6 ttl=64 time=102 ms
64 bytes from 10.0.0.2: icmp_seq=7 ttl=64 time=101 ms
64 bytes from 10.0.0.2: icmp_seq=8 ttl=64 time=102 ms
64 bytes from 10.0.0.2: icmp_seq=9 ttl=64 time=100 ms
^C
--- 10.0.0.2 ping statistics ---
9 packets transmitted, 9 received, 0% packet loss, time 8018ms
rtt min/avg/max/mdev = 100.264/101.240/101.994/0.520 ms
```

Here when the h1 hosts pings h2 or any other host the time taken is consistently above 100ms. However, on pinging any other host on which bottleneck is not introduced we obtain the following result:-

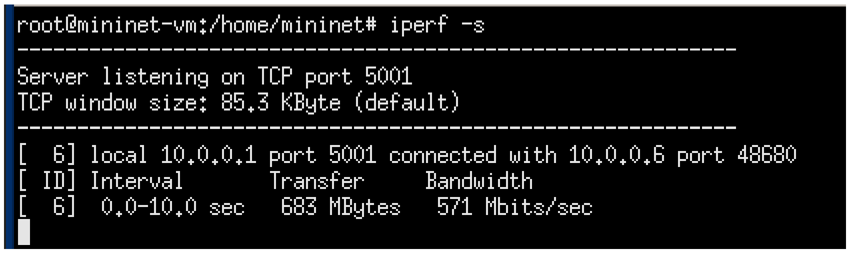


```
mininet> h2 ping  -c 5 h3
PING 10.0.0.3 (10.0.0.3) 56(84) bytes of data.
64 bytes from 10.0.0.3: icmp_seq=1 ttl=64 time=0.182 ms
64 bytes from 10.0.0.3: icmp_seq=2 ttl=64 time=0.086 ms
64 bytes from 10.0.0.3: icmp_seq=3 ttl=64 time=0.101 ms
64 bytes from 10.0.0.3: icmp_seq=4 ttl=64 time=0.106 ms
64 bytes from 10.0.0.3: icmp_seq=5 ttl=64 time=0.101 ms

--- 10.0.0.3 ping statistics ---
5 packets transmitted, 5 received, 0% packet loss, time 4079ms
rtt min/avg/max/mdev = 0.086/0.115/0.182/0.034 ms
```

On pinging a host on which the bottleneck is not introduced, we observe that time take is very less and a delay of 100ms is successfully introduced due to the bottleneck.

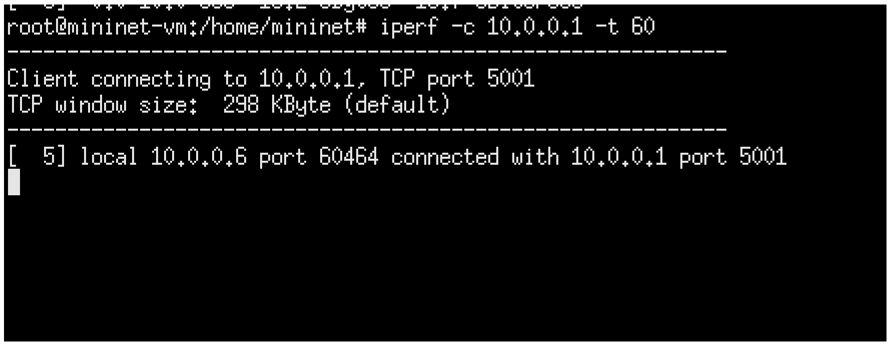
b) Generating tcp traffic between h1 & h6 using iperf & capture the packets generated at host h1.

We first check the ip address of host h1 using ipconfig and obtain it as 10.0.0.1. We then start a server at h1 to accept incoming traffic/connections using iperf command.



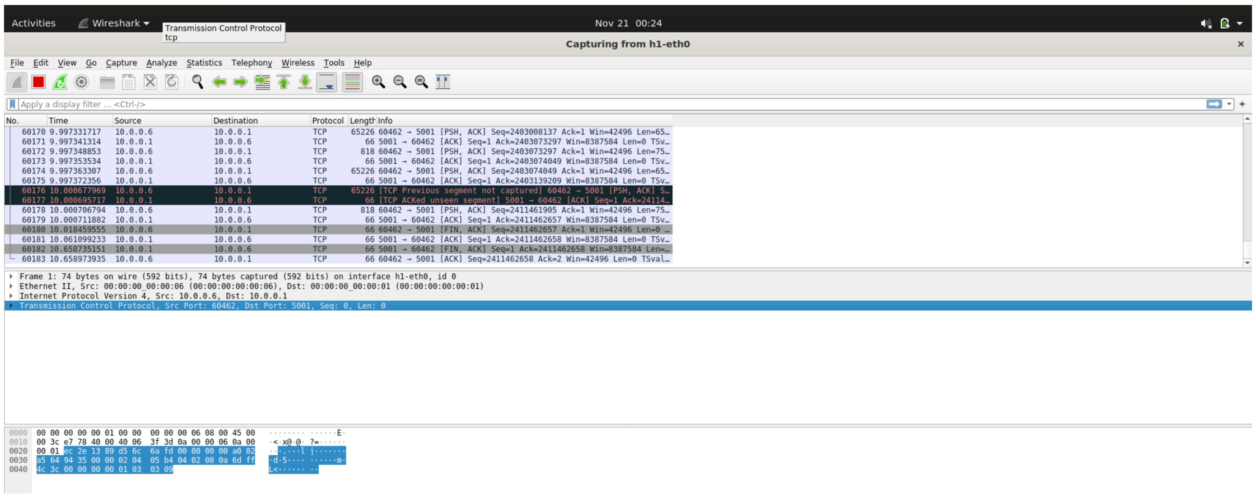
```
root@mininet-vm:/home/mininet# iperf -s
------------------------------------------------------------
Server listening on TCP port 5001
TCP window size: 85.3 KByte (default)
------------------------------------------------------------
[  6] local 10.0.0.1 port 5001 connected with 10.0.0.6 port 48680
[ ID] Interval       Transfer     Bandwidth
[  6]  0.0-10.0 sec   683 MBytes   571 Mbits/sec
```

Now, we set up a client at h6 and make it send packets to h1 and set up wireshark to start capturing the incoming packets at host h1 from h6 and collect the capture as a pcap trace.



```
root@mininet-vm:/home/mininet# iperf -c 10.0.0.1 -t 60
------------------------------------------------------------
Client connecting to 10.0.0.1, TCP port 5001
TCP window size:  298 KByte (default)
------------------------------------------------------------
[  5] local 10.0.0.6 port 60464 connected with 10.0.0.1 port 5001
```

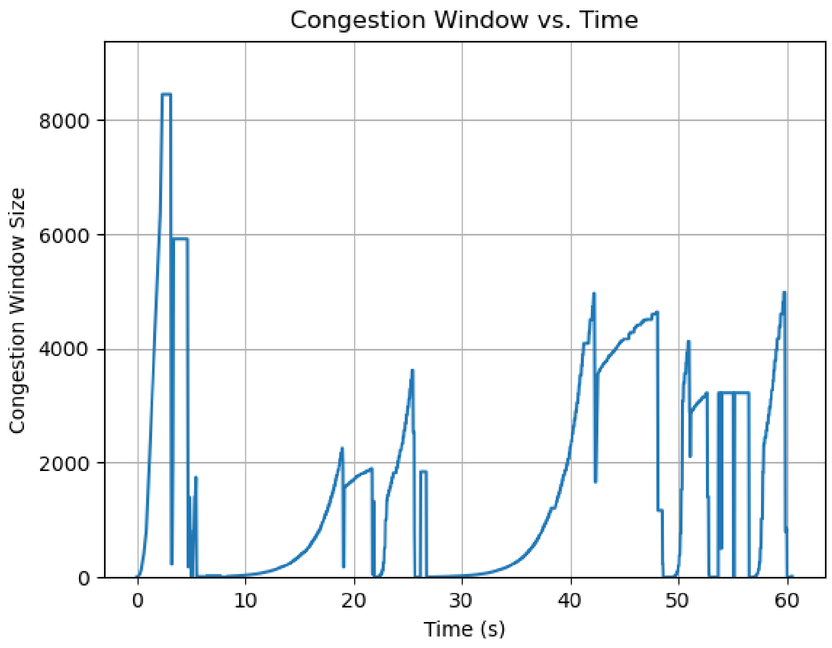
c) Capturing packets at host h1 using wireshark

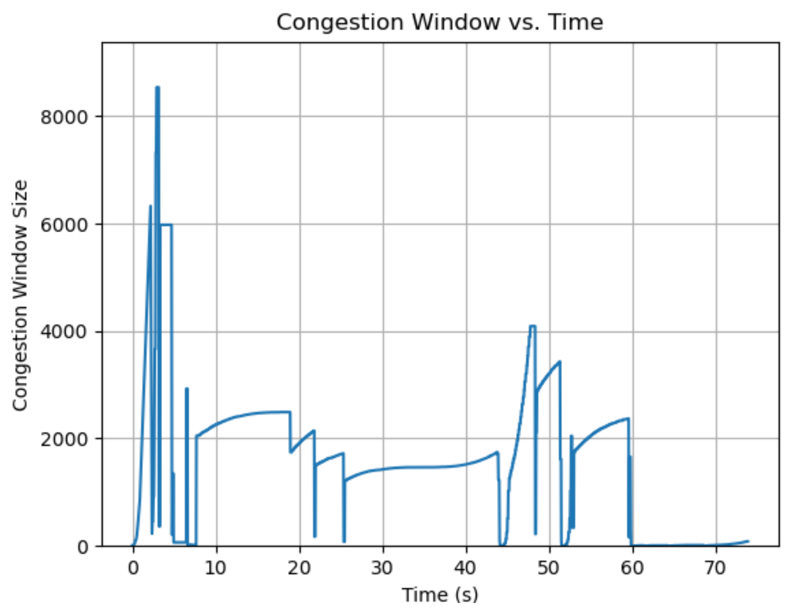

Activities
Wireshark
Transmission Control Protocol
tcp
Nov 21 00:24
Capturing from h1-eth0
File Edit View Go Capture Analyze Statistics Telephony Wireless Tools Help
Apply a display filter ... <Ctrl-/>

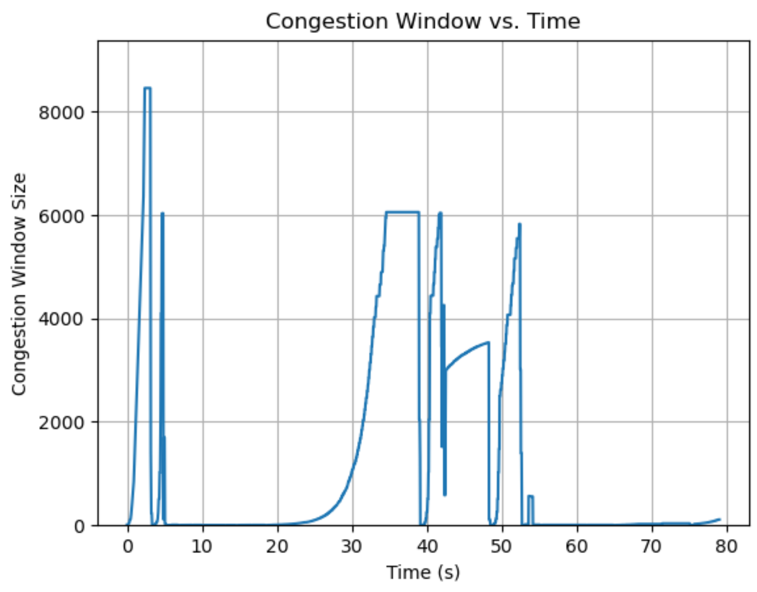
```
No.   Time         Source    Destination Protocol Length Info
60170 9.997331717  10.0.0.6  10.0.0.1    TCP      65226 60462 → 5001 [PSH, ACK] Seq=2403008137 Ack=1 Win=42496 Len=65…
60171 9.997341314  10.0.0.1  10.0.0.6    TCP         66 5001 → 60462 [ACK] Seq=1 Ack=2403073297 Win=8387584 Len=0 TSv…
60172 9.997348853  10.0.0.6  10.0.0.1    TCP        818 60462 → 5001 [PSH, ACK] Seq=2403073297 Ack=1 Win=42496 Len=75…
60173 9.997353534  10.0.0.1  10.0.0.6    TCP         66 5001 → 60462 [ACK] Seq=1 Ack=2403074049 Win=8387584 Len=0 TSv…
60174 9.997363307  10.0.0.6  10.0.0.1    TCP      65226 60462 → 5001 [PSH, ACK] Seq=2403074049 Ack=1 Win=42496 Len=65…
60175 9.997372356  10.0.0.1  10.0.0.6    TCP         66 5001 → 60462 [ACK] Seq=1 Ack=2403139209 Win=8387584 Len=0 TSv…
60176 10.000677969 10.0.0.6  10.0.0.1    TCP      65226 [TCP Previous segment not captured] 60462 → 5001 [PSH, ACK] S…
60177 10.000695717 10.0.0.1  10.0.0.6    TCP         66 [TCP ACKed unseen segment] 5001 → 60462 [ACK] Seq=1 Ack=24114…
60178 10.000706794 10.0.0.6  10.0.0.1    TCP        818 60462 → 5001 [PSH, ACK] Seq=2411461905 Ack=1 Win=42496 Len=75…
60179 10.000711882 10.0.0.1  10.0.0.6    TCP         66 5001 → 60462 [ACK] Seq=1 Ack=2411462657 Win=8387584 Len=0 TSv…
60180 10.018459555 10.0.0.6  10.0.0.1    TCP         66 60462 → 5001 [FIN, ACK] Seq=2411462657 Ack=1 Win=42496 Len=0 …
60181 10.061099233 10.0.0.1  10.0.0.6    TCP         66 5001 → 60462 [ACK] Seq=1 Ack=2411462658 Win=8387584 Len=0 TSv…
60182 10.658735151 10.0.0.1  10.0.0.6    TCP         66 5001 → 60462 [FIN, ACK] Seq=1 Ack=2411462658 Win=8387584 Len=…
60183 10.658973935 10.0.0.6  10.0.0.1    TCP         66 60462 → 5001 [ACK] Seq=2411462658 Ack=2 Win=42496 Len=0 TSval…

▸ Frame 1: 74 bytes on wire (592 bits), 74 bytes captured (592 bits) on interface h1-eth0, id 0
▸ Ethernet II, Src: 00:00:00_00:00:06 (00:00:00:00:00:06), Dst: 00:00:00_00:00:01 (00:00:00:00:00:01)
▸ Internet Protocol Version 4, Src: 10.0.0.6, Dst: 10.0.0.1
▸ Transmission Control Protocol, Src Port: 60462, Dst Port: 5001, Seq: 0, Len: 0

0000  00 00 00 00 00 01 00 00  00 00 00 06 08 00 45 00   ········ ······E·
0010  00 3c e7 78 40 00 40 06  3f 3d 0a 00 00 06 0a 00   ·<·x@·@· ?=······
0020  00 01 ec 2e 13 89 d5 6c  6a fd 00 00 00 00 a0 02   ···.···l j·······
0030  a5 64 94 35 00 00 02 04  05 b4 04 02 08 0a 6d ff   ·d·5···· ······m·
0040  4c 3c 00 00 00 00 01 03  03 09                     L<······ ··
```

d) These are some of the plots for the congestion window size over time. The plots were generated using the ss tool in linux which is used to get the socket statistics at the tcp sender. We cannot use the pcaps to get information about congestion window size since they are not included in any packet header as these are used only on the sender side.

A bash script as demonstrated in the tutorial was used to get the congestion window and other related stats in the form of a .csv file. The csv file was then read using python scripts to plot the graph between congestion window and time.







Observations:

- The plots obey the tcp slow start mechanism and then gradually increase the amount of data being sent.
- Once the value ss-threshold value is reached it goes into the congestion avoidance phase and the congestion window grows slowly now.
- Also we can see the tcp sawtooth behavior.

References

https://witestlab.poly.edu/blog/tcp-congestion-control-basics/

https://raw.githubusercontent.com/ffund/tcp-ip-essentials/gh-pages/scripts/ss-output.sh

https://www.brianlinkletter.com/2015/04/using-the-pox-sdn-controller/

https://www.youtube.com/watch?v=_WgUwUf1d34&ab_channel=DavidMahler

https://ervikrant06.wordpress.com/2015/09/17/learning-ovs-open-vswitch-using-mininet-part-1/