

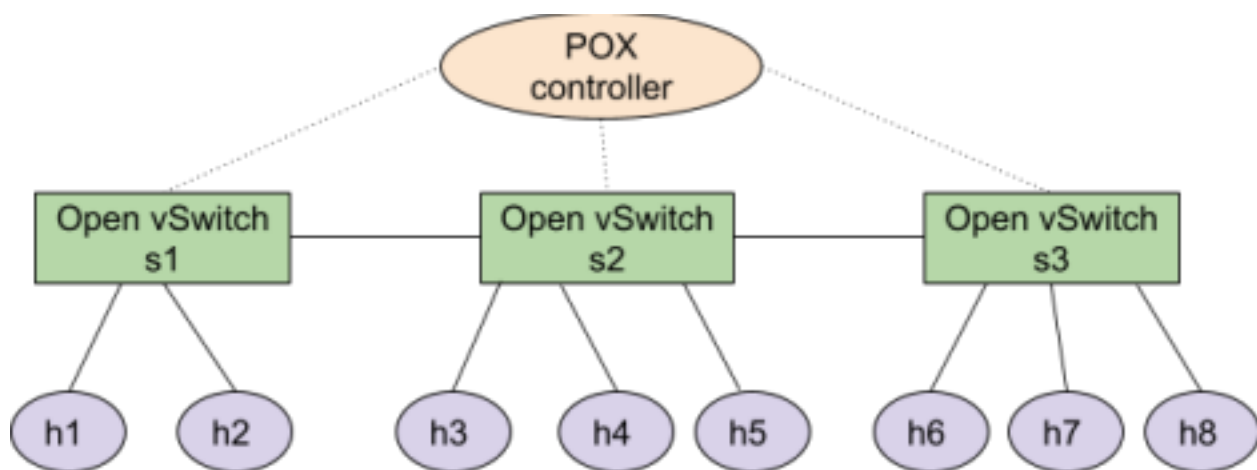
Assignment 4

Understanding network emulation software, Mininet

Q.1.

A.

Three functions are used to make a custom network topology as shown in figure below:



`addHost()`, `addSwitch()` and `addLink()` are those three functions.

`addHost()` is used to add hosts, `addLink()` is used to add links between switches and hosts and add links between switches, and `addSwitch` is used to add switches.

```
def build( self ):
    "Create custom topo."

    # Add hosts and switches

    Swth1 = self.addSwitch( 's1' )
    Swth2 = self.addSwitch( 's2' )
    Swth3 = self.addSwitch( 's3' )
    Hos1 = self.addHost( 'h1' )
    Hos2 = self.addHost( 'h2' )
    Hos3 = self.addHost( 'h3' )
    Hos4 = self.addHost( 'h4' )
    Hos5 = self.addHost( 'h5' )
    Hos6 = self.addHost( 'h6' )
    Hos7 = self.addHost( 'h7' )
    Hos8 = self.addHost( 'h8' )
    # Add links
    self.addLink( Hos1, Swth1 )
    self.addLink( Hos2, Swth1 )
    self.addLink( Hos3, Swth2 )
    self.addLink( Hos4, Swth2 )
    self.addLink( Hos5, Swth2 )
    self.addLink( Hos6, Swth3 )
    self.addLink( Hos7, Swth3 )
    self.addLink( Hos8, Swth3 )
    self.addLink( Swth1, Swth2 )
    self.addLink( Swth2, Swth3 )
```

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

```

mininet@mininet-vm:~/mininet/custom$ sudo mn --custom topo-2sw-2host.py --topo m
ytopo
*** Creating network
*** Adding controller
*** Adding hosts:
h1 h2 h3 h4 h5 h6 h7 h8
*** Adding switches:
s3 s4 s5
*** Adding links:
(h1, s3) (h2, s3) (h3, s4) (h4, s4) (h5, s4) (h6, s5) (h7, s5) (h8, s5) (s3, s4)
(s4, s5)
*** Configuring hosts
h1 h2 h3 h4 h5 h6 h7 h8
*** Starting controller
c0
*** Starting 3 switches
s3 s4 s5 ...
*** Starting CLI:
mininet> █

```

B.

The command "net" is quite powerful in Mininet and is used to manage a variety of network-related tasks. There are many functions accessible, including network configuration, network share management, user and group administration, etc.

```

mininet> net
h1 h1-eth0:s3-eth1
h2 h2-eth0:s3-eth2
h3 h3-eth0:s4-eth1
h4 h4-eth0:s4-eth2
h5 h5-eth0:s4-eth3
h6 h6-eth0:s5-eth1
h7 h7-eth0:s5-eth2
h8 h8-eth0:s5-eth3
s3 lo: s3-eth1:h1-eth0 s3-eth2:h2-eth0 s3-eth3:s4-eth4
s4 lo: s4-eth1:h3-eth0 s4-eth2:h4-eth0 s4-eth3:h5-eth0 s4-eth4:s3-eth3 s4-eth5:s5-eth4
s5 lo: s5-eth1:h6-eth0 s5-eth2:h7-eth0 s5-eth3:h8-eth0 s5-eth4:s4-eth5
c0
mininet> █

```

Q.2.

a.

To display the network traffic control (tc) queuing discipline (qdisc) configuration for the network interface h1 -eth0 on the host h1, I used the command 'h1 tc qdisc show dev h1 -eth0'. Then, to make the bandwidth 1 Mbps, Burst 25 Kbit and latency 400 ms, I used the command 'h1 tc qdisc add dev h1-eth0 root tbf rate 1mbit burst 25kbit latency 400ms'.

Lastly, to show the updated values of bandwidth, burst and latency I again used the command 'h1 tc qdisc show dev h1-eth0'.

```
mininet> h1 tc qdisc show dev h1-eth0
mininet> h1 tc qdisc add dev h1-eth0 root tbf rate 1mbit burst 25kbit latency 400ms
mininet> h1 tc qdisc show dev h1-eth0
qdisc tbf 8003: root refcnt 2 rate 1Mbit burst 3200b lat 400.0ms
mininet> █
```

b.

To start iperf in server mode, listening on port 5001, I ran the command 'h6 iperf -s -p 5001 &'. We set the h6 as a server using the previous command. Then, to start iperf in client mode, listening on port 5001, I ran the command 'h1 iperf -c h6 -p 5001'. We set the h6 as the client and establish a TCP connection with h6 using the previous command. Finally, the following output shows the local IP address, TCP window size, the client connecting to h6, the bandwidth achieved during the test and the transfer interval.

```
mininet> h6 iperf -s -p 5001 &
mininet> h1 iperf -c h6 -p 5001
-----
Client connecting to 10.0.0.6, TCP port 5001
TCP window size: 85.3 KByte (default)
-----
[  3] local 10.0.0.1 port 46868 connected with 10.0.0.6 port 5001
[ ID] Interval      Transfer    Bandwidth
[  3]  0.0-10.0 sec  28.9 GBytes  24.9 Gbits/sec
mininet> █
```

In the above image, a transfer of 28.9 GigaBytes was achieved at a bandwidth of 24.9 Gigabits per sec.

c.

I have used wireshark to capture the packets, Below is the screenshot of the wireshark. Along with this I have submitted a .pcap file.

traffic.pcap						
File Edit View Go Capture Analyze Statistics Telephony Wireless Tools Help						
Apply a display filter ... <Ctrl-/> Expression... +						
No.	Time	Source	Destination	Protocol	Length	Info
15138	10.002890199	10.0.0.1	10.0.0.6	TCP	65226	54004 → 5001 [ACK] Seq=2586722897 Ac...
15139	10.002892205	10.0.0.1	10.0.0.6	TCP	65226	54004 → 5001 [ACK] Seq=2586788057 Ac...
15140	10.002896457	10.0.0.1	10.0.0.6	TCP	49298	54004 → 5001 [ACK] Seq=2586853217 Ac...
15141	10.002900513	10.0.0.1	10.0.0.6	TCP	15994	54004 → 5001 [ACK] Seq=2586902449 Ac...
15142	10.002902342	10.0.0.1	10.0.0.6	TCP	65226	54004 → 5001 [ACK] Seq=2586918377 Ac...
15143	10.002905706	10.0.0.1	10.0.0.6	TCP	65226	54004 → 5001 [ACK] Seq=2586983537 Ac...
15144	10.002908744	10.0.0.1	10.0.0.6	TCP	65226	54004 → 5001 [ACK] Seq=2587048697 Ac...
15145	10.002910033	10.0.0.1	10.0.0.6	TCP	65226	54004 → 5001 [ACK] Seq=2587113857 Ac...
15146	10.002912486	10.0.0.1	10.0.0.6	TCP	65226	54004 → 5001 [ACK] Seq=2587179017 Ac...
15147	11.691410598	10.0.0.1	10.0.0.6	TCP	66	[TCP Previous segment not captured] ...
15148	11.697004816	10.0.0.6	10.0.0.1	TCP	66	[TCP ACKed unseen segment] 5001 → 54...
15149	11.697020361	10.0.0.1	10.0.0.6	TCP	66	54004 → 5001 [ACK] Seq=2618032154 Ac...
▶ Frame 15144: 65226 bytes on wire (521808 bits), 65226 bytes captured (521808 bits) on interface 0 ▶ Ethernet II, Src: 4a:16:b0:8a:1f:4b (4a:16:b0:8a:1f:4b), Dst: 7a:e6:65:a3:5b:03 (7a:e6:65:a3:5b:03) ▶ Internet Protocol Version 4, Src: 10.0.0.1, Dst: 10.0.0.6 ▶ Transmission Control Protocol, Src Port: 54004, Dst Port: 5001, Seq: 2587048697, Ack: 1, Len: 65160 ▶ Data (65160 bytes)						
0000	7a e6 65 a3 5b 03 4a 16 b0 8a 1f 4b 08 00 45 00	z e [. J . . . K . E .				
0010	fe bc b5 5e 40 00 40 06 72 d6 0a 00 00 01 0a 00	. . . ^ @ . r				
0020	00 06 d2 f4 13 89 f7 5b 88 78 95 25 00 ee 80 10 [. x . % . . .				
0030	00 3a 12 b6 00 00 01 01 08 0a 00 00 69 cf 00 00	. : i				
0040	69 be 38 39 30 31 32 33 34 35 36 37 38 39 30 31	i . 890123 45678901				
0050	32 33 34 35 36 37 38 39 30 31 32 33 34 35 36 37	23456789 01234567				
0060	38 39 30 31 32 33 34 35 36 37 38 39 30 31 32 33	89012345 67890123				
0070	34 35 36 37 38 39 30 31 32 33 34 35 36 37 38 39	45678901 23456789				
0080	30 31 32 33 34 35 36 37 38 39 30 31 32 33 34 35	01234567 89012345				
0090	36 37 38 39 30 31 32 33 34 35 36 37 38 39 30 31	67890123 45678901				
00a0	32 33 34 35 36 37 38 39 30 31 32 33 34 35 36 37	23456789 01234567				
00b0	38 39 30 31 32 33 34 35 36 37 38 39 30 31 32 33	89012345 67890123				
00c0	34 35 36 37 38 39 30 31 32 33 34 35 36 37 38 39	45678901 23456789				

d.

I have used the .pcap file to generate a graph plot using a python script. In the graph, the x-axis represents the time(in sec) and the y-axis represents the congestion window size. The size of the congestion window is equal to the number of packets the sender can transmit before an acknowledgement from the receiver is needed. The graph details changes to this congestion window size throughout the interaction between the client and server. This assessment helps us grasp the network's controls for managing congestion and how effective the process of data transfer is. It may provide understanding about how the network performs, where the traffic is heavy, possible enhancements to the network infrastructure and bottlenecks.

Graph plot of Congestion Window Over Time is given below:

