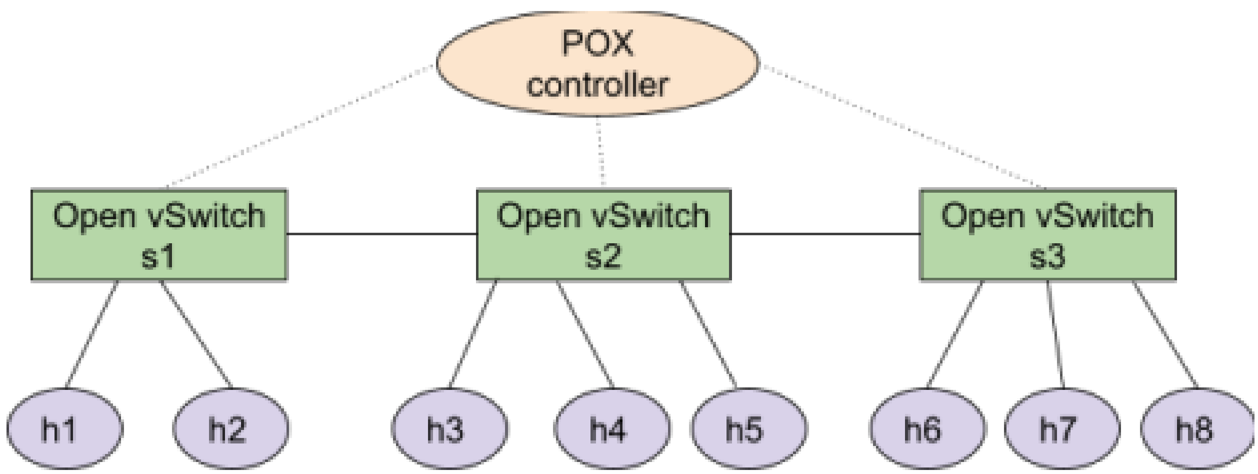
Assignment 4
Understanding network emulation software, Mininet

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 ‘hl tc qdisc add dev hl-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.

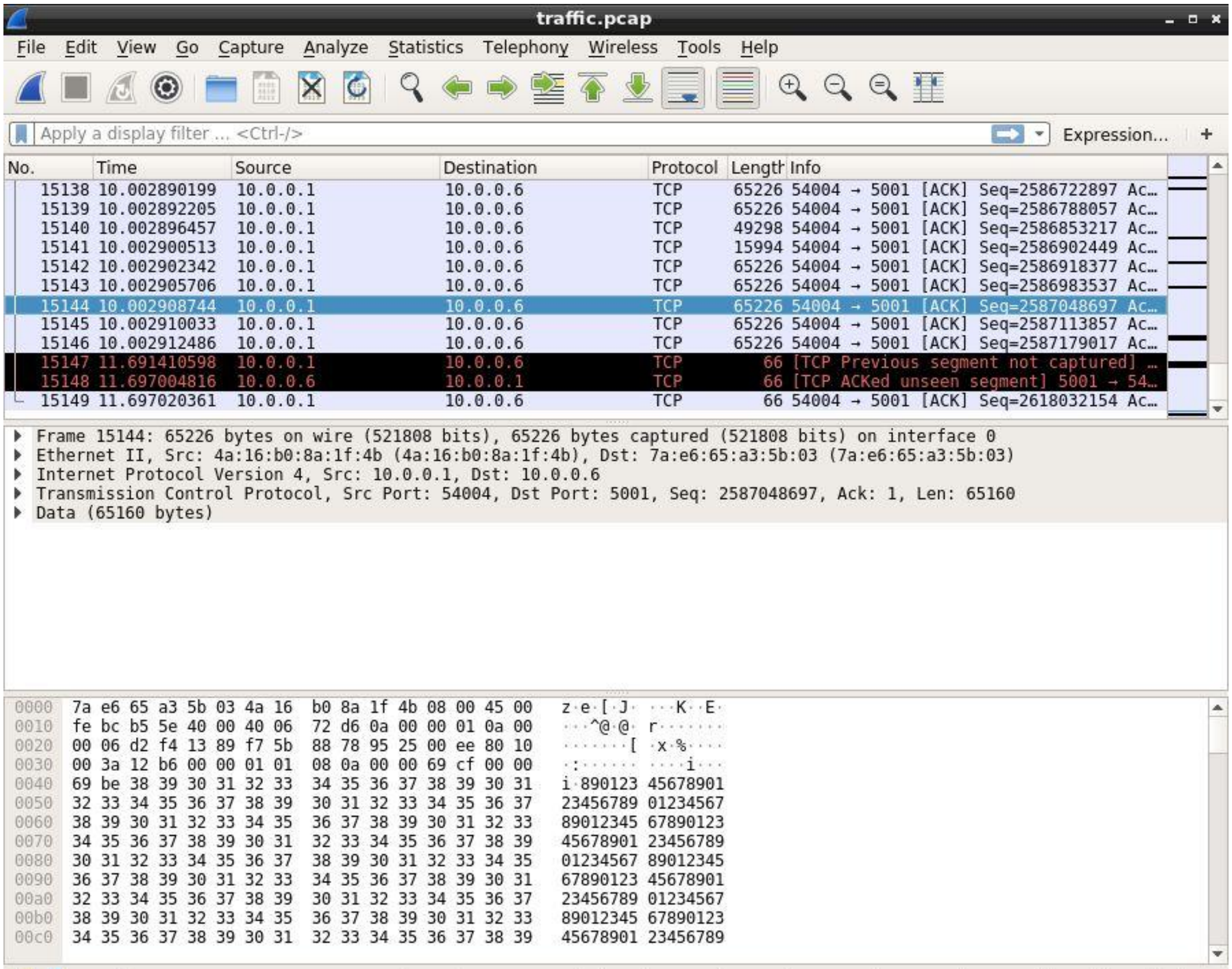
To start iperf in server mode, listening on port 5001, I ran the command ‘h6 iperf -s -p 5001 &’. We set the h6 as a server using the previous command. Then, to start iperf in client mode, listening on port 5001, I ran the command ‘h1 iperf -c h6 -p 5001’. We set the h6 as the client and establish a 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.

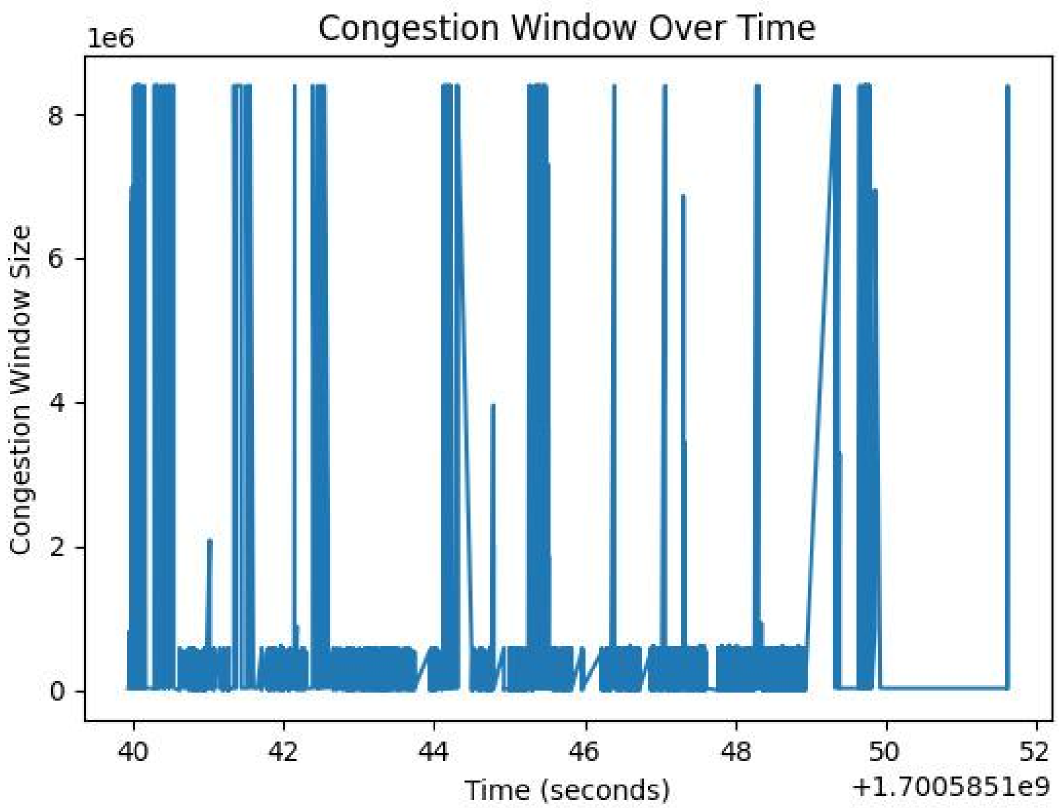
I have used wireshark to capture the packets, Below is the screenshot of the wireshark. Along with this I have submitted a .pcap file.



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:



Congestion Window Over Time
1e6
8
6
4
2
0
Congestion Window Size
40
42
44
46
48
50
52
Time (seconds)
+1.7005851e9