Computer Networks

Assignment 4

Question1:

A

i. After following the tutorial, I created the topology given in Question 2.

ii. After writing topology

topo-2sw-2host.py

```
from mininet.topo import Topo

class MyTopo(Topo):
    "Simple topology example."

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

        # Add switches
        s1 = self.addSwitch('s1')
        s2 = self.addSwitch('s2')
        s3 = self.addSwitch('s3')

        # Add hosts
        h1 = self.addHost('h1')
        h2 = self.addHost('h2')
        h3 = self.addHost('h3')
        h4 = self.addHost('h4')
        h5 = self.addHost('h5')
        h6 = self.addHost('h6')
        h7 = self.addHost('h7')
        h8 = self.addHost('h8')

        # Add links
        self.addLink(s1, h1)
        self.addLink(s1, h2)
        self.addLink(s2, h3)
        self.addLink(s2, h4)
        self.addLink(s2, h5)
        self.addLink(s3, h6)
        self.addLink(s3, h7)
        self.addLink(s3, h8)
        self.addLink(s1, s2)
        self.addLink(s2, s3)

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

For the POX controller, we need to activate POX controller:-

The POX controller is a networking software platform written in Python. It can be used to implement the OpenFlow protocol, which is the standard communication protocol between switches and controllers.

Using command :-

sudo ~/pox/pox.py forwarding.l2_learning \ openflow.spanning_tree --no-flood --hold-down \ log.level --DEBUG samples.pretty_log \ openflow.discovery host_tracker \ info.packet_dump

Running command

$ sudo mn --custom ~/mininet/custom/topo-2sw-2host.py --topo mytopo –mac --switch ovsk --controller remote

Yogender Kumar 2021505

```
yogender@yogender-VirtualBox:~/mininet$ sudo mn --custom ~/mininet/custom/topo-2sw-2host.py --top
o mytopo --mac --switch ovsk --controller remote
*** Creating network
*** Adding controller
Unable to contact the remote controller at 127.0.0.1:6653
Connecting to remote controller at 127.0.0.1:6633
*** Adding hosts:
h1 h2 h3 h4 h5 h6 h7 h8
*** Adding switches:
s1 s2 s3
*** Adding links:
(s1, h1) (s1, h2) (s1, s2) (s2, h3) (s2, h4) (s2, h5) (s2, s3) (s3, h6) (s3, h7) (s3, h8)
*** Configuring hosts
h1 h2 h3 h4 h5 h6 h7 h8
*** Starting controller
c0
*** Starting 3 switches
s1 s2 s3 ...
*** Starting CLI:
mininet>
```

B.

Verifying topology by checking the network configuration using command:

```
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>
```

Question 2:

a.

Here, I am creating a limit for transfer rate as bottleneck , which gives the variation between bandwidth.

Using command :

$ h1 tc qdisc add dev h1-eth0 root tbf rate 1mbit burst 1000kbit latency 10ms

Here, we set the transfer rate 1024kb ~= 1mbit.

```
mininet> iperf
*** Iperf: testing TCP bandwidth between h1 and h8
*** Results: ['17.9 Gbits/sec', '17.9 Gbits/sec']
mininet> h1 tc qdisc add dev h1-eth0 root tbf rate 1mbit burst 1000kbit latency 10ms
mininet> iperf
*** Iperf: testing TCP bandwidth between h1 and h8
*** Results: ['1.0 Mbits/sec', '1.5 Mbits/sec']
mininet>
```

Observation:

Before running command :

The TCP bandwidth between h1 and h8 nodes is ’17.9 Gbits/sec’ as shown above.

After running command :

The TCP bandwidth between h1 and h8 nodes is ’1.0 mbits/sec and 1.5mbits/sec’ as shown above.

b.

Using command:

$ iperf h1 h6

```
mininet> iperf h1 h6
*** Iperf: testing TCP bandwidth between h1 and h6
*** Results: ['1.0 Mbits/sec', '1.4 Mbits/sec']
mininet>
```

Observation:

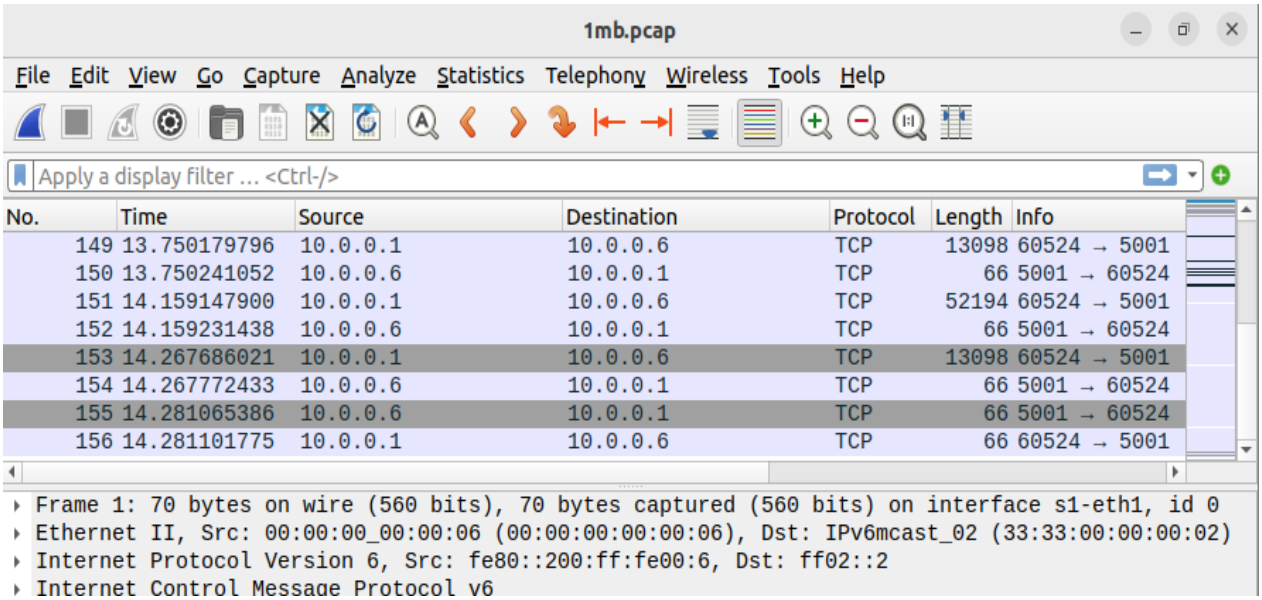
Using “iperf h1 h6” measures the bandwidth between the two nodes.

c.

Using Wireshark we capture the Packets from (b) part.

By running command:

$ sudo wireshark

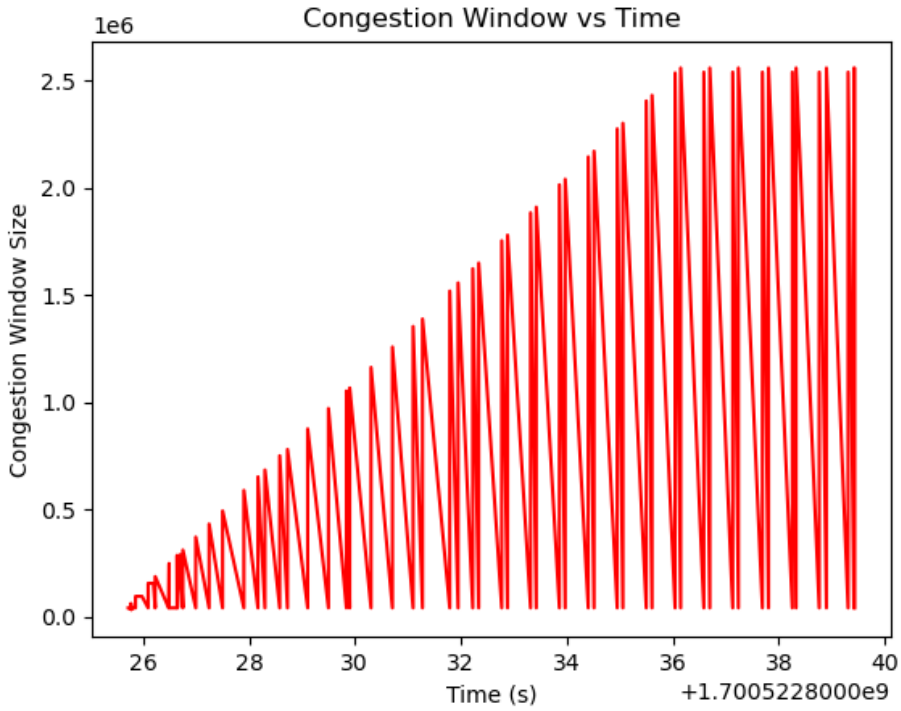


Observation:

Here , h1 is client (10.0.0.1) ip address where as h6 is server (10.0.0.6) ip address, using TCP.

d.

Plot between congestion window(segment) with time(seconds)



Observation:

The bandwidth has been artificially fixed using the tc command, which means the network is constrained to a specific data rate. Even with a fixed bandwidth using tc, the TCP slow start phase can still be observed as the sender adapts to the artificial constraint. The linear growth in the congestion window size indicates that the sender is efficiently exploring the available bandwidth without causing congestion.

Note: - Python script for plot is in directory.

References

https://www.youtube.com/watch?v=r52F5_j7nLY

https://www.brianlinkletter.com/2015/09/using-pox-components-to-create-a-software-defined-networking-application/#:~:text=In%20the%20new%20terminal%20window%20%28or%20tab%29%2C%20we,log.level%20--DEBUG%20samples.pretty_log%20%20openflow.discovery%20host_tracker%20%20info.packet_dump

Tutorial 5 and 7

https://www.youtube.com/watch?v=Kp1OSorEJ78&ab_channel=RKiLAB