Networks Lab Report
Assignment 2

AIM of the Assignment : The objective of this assignment is to understand the Wireshark tool and how you can analyse network packet traces. The experiments need to be executed over the Mininet environment. The assignment statements are almost similar to what we have done in Assignment 1, but this time we have to use Wireshark for answering the questions. So primarily we use wireshark to visually see the packet sent and received through UDP and TCP protocols and justify the various attributes of the protocols such as packet loss, delay through the amount of packets sent received and acknowledged.

Part 1

Procedure

Assuming mininet is set up on the system in a virtualbox, first boot up mininet from the oracle vm virtual box, now type the following and note the ip for eth0.

```
>ifconfig
```

Now, ssh login to your virtual mininet using local terminal (to create xterms for h1 and h2), use ip you got from previous step from mininet terminal. And type the following for desired topology for part 1:

```
>sudo mn --link tc,bw=1,delay=1ms,loss=0
```

```
*** Creating network
*** Adding controller
*** Adding hosts:
h1 h2
*** Adding switches:
s1
*** Adding links:
(1.00Mbit 1ms delay 0% loss) (1.00Mbit 1ms delay 0% loss) (h1, s1) (1.00Mbit 1ms
 delay 0% loss) (1.00Mbit 1ms delay 0% loss) (h2, s1)
*** Configuring hosts
h1 h2
*** Starting controller
c0
*** Starting 1 switches
s1 ...(1.00Mbit 1ms delay 0% loss) (1.00Mbit 1ms delay 0% loss)
```

Setting bandwidth to 1Mbps, delay to 1ms and loss to 1% for left link and 2% for right link

Now, from your terminal use

>xterm h2

And type the following to capture packets from h2

>h2 sudo tcpdump -n -t > h2_tcp &

And enter the following command for creating TCP server in H2

>Iperf -s

```
root@mininet-vm:~# iperf -s
------------------------------------------------------------
Server listening on TCP port 5001
TCP window size: 85.3 KByte (default)
------------------------------------------------------------
[ 14] local 10.0.0.2 port 5001 connected with 10.0.0.1 port 56484
[ ID] Interval       Transfer     Bandwidth
[ 14]  0.0-19.7 sec  2.25 MBytes   957 Kbits/sec
```

This is now our TCP server.

Now open the client by typing the following in your terminal

>xterm h1

And type the following to capture packets from h1

>h1 sudo tcpdump -n -t > h1_tcp &

And type the following for TCP throughput test

>iperf -c 10.0.0.2

```
root@mininet-vm:~# iperf -c 10.0.0.2
------------------------------------------------------------
Client connecting to 10.0.0.2, TCP port 5001
TCP window size: 85.3 KByte (default)
------------------------------------------------------------
[ 13] local 10.0.0.1 port 56568 connected with 10.0.0.2 port 5001
[ ID] Interval       Transfer     Bandwidth
[ 13]  0.0-10.6 sec  2.12 MBytes  1.69 Mbits/sec
```

This is the xterm window of H2.

```
root@mininet-vm:~# iperf -s
------------------------------------------------------------
Server listening on TCP port 5001
TCP window size: 85.3 KByte (default)
------------------------------------------------------------
[ 14] local 10.0.0.2 port 5001 connected with 10.0.0.1 port 56568
[ ID] Interval       Transfer     Bandwidth
[ 14]  0.0-18.6 sec  2.12 MBytes   958 Kbits/sec
```

Example of TCP capture output of h1 :

```
ARP, Request who-has 10.0.0.2 tell 10.0.0.1, length 28
ARP, Reply 10.0.0.2 is-at 36:a6:14:ff:4d:ae, length 28
IP 10.0.0.1.56524 > 10.0.0.2.5001: Flags [S], seq 2379991754, win 29200, options [mss 1460,sackOK,TS val 10941121 ecr 0,nop,wscale 9], length
IP 10.0.0.2.5001 > 10.0.0.1.56524: Flags [S.], seq 1031466246, ack 2379991755, win 28960, options [mss 1460,sackOK,TS val 10941123 ecr 109411
nop,wscale 9], length 0
IP 10.0.0.1.56524 > 10.0.0.2.5001: Flags [.], ack 1, win 58, options [nop,nop,TS val 10941124 ecr 10941123], length 0
IP 10.0.0.1.56524 > 10.0.0.2.5001: Flags [P.], seq 1:25, ack 1, win 58, options [nop,nop,TS val 10941124 ecr 10941123], length 24
IP 10.0.0.1.56524 > 10.0.0.2.5001: Flags [.], seq 25:2921, ack 1, win 58, options [nop,nop,TS val 10941124 ecr 10941123], length 2896
IP 10.0.0.2.5001 > 10.0.0.1.56524: Flags [.], ack 25, win 57, options [nop,nop,TS val 10941124 ecr 10941124], length 0
IP 10.0.0.2.5001 > 10.0.0.1.56524: Flags [.], ack 1473, win 63, options [nop,nop,TS val 10941124 ecr 10941124], length 0
IP 10.0.0.2.5001 > 10.0.0.1.56524: Flags [.], ack 2921, win 68, options [nop,nop,TS val 10941125 ecr 10941124], length 0
IP 10.0.0.1.56524 > 10.0.0.2.5001: Flags [.], seq 2921:5817, ack 1, win 58, options [nop,nop,TS val 10941124 ecr 10941123], length 2896
IP 10.0.0.2.5001 > 10.0.0.1.56524: Flags [.], ack 5817, win 80, options [nop,nop,TS val 10941128 ecr 10941124], length 0
IP 10.0.0.1.56524 > 10.0.0.2.5001: Flags [.], seq 5817:8713, ack 1, win 58, options [nop,nop,TS val 10941124 ecr 10941123], length 2896
IP 10.0.0.2.5001 > 10.0.0.1.56524: Flags [.], ack 8713, win 91, options [nop,nop,TS val 10941134 ecr 10941124], length 0
IP 10.0.0.1.56524 > 10.0.0.2.5001: Flags [.], seq 8713:11609, ack 1, win 58, options [nop,nop,TS val 10941124 ecr 10941123], length 2896
IP 10.0.0.2.5001 > 10.0.0.1.56524: Flags [.], ack 11609, win 102, options [nop,nop,TS val 10941140 ecr 10941124], length 0
IP 10.0.0.1.56524 > 10.0.0.2.5001: Flags [.], seq 11609:13057, ack 1, win 58, options [nop,nop,TS val 10941124 ecr 10941123], length 1448
IP 10.0.0.2.5001 > 10.0.0.1.56524: Flags [.], ack 13057, win 108, options [nop,nop,TS val 10941146 ecr 10941124], length 0
IP 10.0.0.1.56524 > 10.0.0.2.5001: Flags [P.], seq 13057:15953, ack 1, win 58, options [nop,nop,TS val 10941125 ecr 10941124], length 2896
IP 10.0.0.2.5001 > 10.0.0.1.56524: Flags [.], ack 15953, win 119, options [nop,nop,TS val 10941149 ecr 10941125], length 0
IP 10.0.0.1.56524 > 10.0.0.2.5001: Flags [.], seq 15953:18849, ack 1, win 58, options [nop,nop,TS val 10941125 ecr 10941124], length 2896
IP 10.0.0.2.5001 > 10.0.0.1.56524: Flags [.], ack 18849, win 131, options [nop,nop,TS val 10941155 ecr 10941125], length 0
IP 10.0.0.1.56524 > 10.0.0.2.5001: Flags [.], seq 18849:21745, ack 1, win 58, options [nop,nop,TS val 10941125 ecr 10941125], length 2896
IP 10.0.0.2.5001 > 10.0.0.1.56524: Flags [.], ack 21745, win 142, options [nop,nop,TS val 10941161 ecr 10941125], length 0
IP 10.0.0.1.56524 > 10.0.0.2.5001: Flags [.], seq 21745:24641, ack 1, win 58, options [nop,nop,TS val 10941128 ecr 10941128], length 2896
IP 10.0.0.2.5001 > 10.0.0.1.56524: Flags [.], ack 24641, win 153, options [nop,nop,TS val 10941167 ecr 10941128], length 0
IP 10.0.0.1.56524 > 10.0.0.2.5001: Flags [.], seq 24641:27537, ack 1, win 58, options [nop,nop,TS val 10941128 ecr 10941128], length 2896
IP 10.0.0.2.5001 > 10.0.0.1.56524: Flags [.], ack 27537, win 165, options [nop,nop,TS val 10941173 ecr 10941128], length 0
IP 10.0.0.1.56524 > 10.0.0.2.5001: Flags [.], seq 27537:28985, ack 1, win 58, options [nop,nop,TS val 10941132 ecr 10941128], length 1448
IP 10.0.0.2.5001 > 10.0.0.1.56524: Flags [.], ack 28985, win 170, options [nop,nop,TS val 10941179 ecr 10941132], length 0
IP 10.0.0.1.56524 > 10.0.0.2.5001: Flags [.], seq 28985:30433, ack 1, win 58, options [nop,nop,TS val 10941134 ecr 10941134], length 1448
IP 10.0.0.2.5001 > 10.0.0.1.56524: Flags [.], ack 30433, win 176, options [nop,nop,TS val 10941182 ecr 10941134], length 0
IP 10.0.0.1.56524 > 10.0.0.2.5001: Flags [P.], seq 30433:33329, ack 1, win 58, options [nop,nop,TS val 10941134 ecr 10941134], length 2896
IP 10.0.0.2.5001 > 10.0.0.1.56524: Flags [.], ack 33329, win 187, options [nop,nop,TS val 10941185 ecr 10941134], length 0
IP 10.0.0.1.56524 > 10.0.0.2.5001: Flags [.], seq 33329:34777, ack 1, win 58, options [nop,nop,TS val 10941140 ecr 10941134], length 1448
IP 10.0.0.2.5001 > 10.0.0.1.56524: Flags [.], ack 34777, win 193, options [nop,nop,TS val 10941191 ecr 10941140], length 0
IP 10.0.0.1.56524 > 10.0.0.2.5001: Flags [.], seq 34777:36225, ack 1, win 58, options [nop,nop,TS val 10941140 ecr 10941140], length 1448
IP 10.0.0.2.5001 > 10.0.0.1.56524: Flags [.], ack 36225, win 198, options [nop,nop,TS val 10941194 ecr 10941140], length 0
```

II. Follow the steps in I of part1 above with just the following modification :

a.For all iperf commands (in xterm of H1 and H2) add a '-u' flag and for iperf of client (H1's xterm window) add a '-b <specified bandwidth>' flag too. That is:

In xterm of H2 ,type:

>iperf -s -u

Instead of

>iperf -s

```
root@mininet-vm:~# iperf -s -u
------------------------------------------------------------
Server listening on UDP port 5001
Receiving 1470 byte datagrams
UDP buffer size:  208 KByte (default)
------------------------------------------------------------
```

Now, H2 becomes our UDP server.

And in xterm of H1, type (for part 1.2.1):

>iperf -c 10.0.0.2 -u -b 64k

Instead of

>iperf -c 10.0.0.2

```
root@mininet-vm:~# iperf -c 10.0.0.2 -u -b 64k
------------------------------------------------------------
Client connecting to 10.0.0.2, UDP port 5001
Sending 1470 byte datagrams
UDP buffer size:  208 KByte (default)
------------------------------------------------------------
[ 13] local 10.0.0.1 port 40019 connected with 10.0.0.2 port 5001
[ ID] Interval       Transfer     Bandwidth
[ 13]  0.0-10.3 sec  80.4 KBytes  64.0 Kbits/sec
[ 13] Sent 56 datagrams
[ 13] Server Report:
[ 13]  0.0-10.3 sec  80.4 KBytes  64.0 Kbits/sec   0.140 ms    0/   56 (0%)
```

Now just like for TCP ,we get the required UDP throughput reading from the bandwidth of the UDP server that is the xterm window of H2.

```
root@mininet-vm:~# iperf -s -u
------------------------------------------------------------
Server listening on UDP port 5001
Receiving 1470 byte datagrams
UDP buffer size:  208 KByte (default)
------------------------------------------------------------
[ 13] local 10.0.0.2 port 5001 connected with 10.0.0.1 port 40019
[ ID] Interval       Transfer     Bandwidth        Jitter   Lost/Total Datagrams
[ 13]  0.0-10.3 sec  80.4 KBytes  64.0 Kbits/sec   0.140 ms    0/   56 (0%)
```

Rest tcpdump commands remain the same.

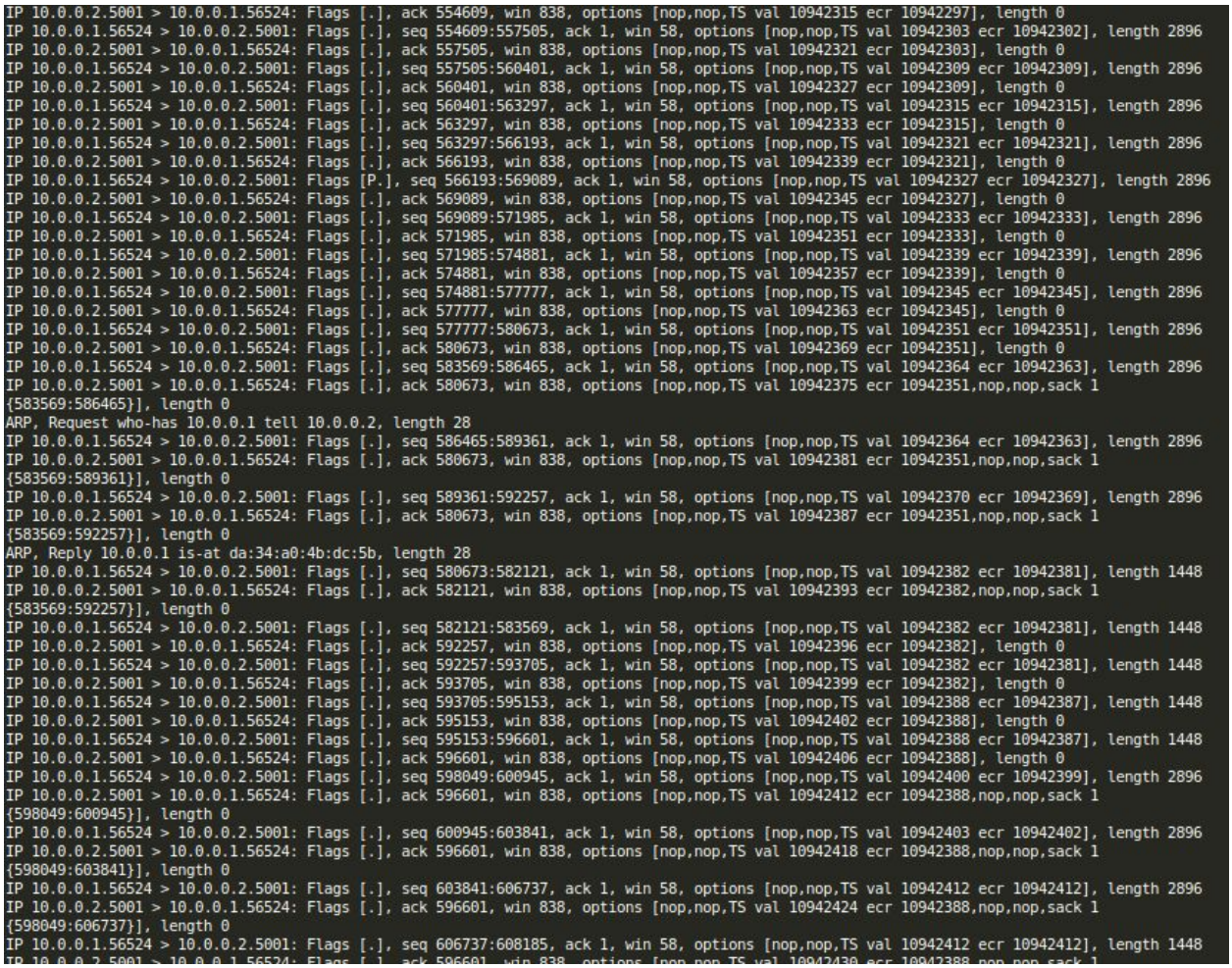
(For UDP, h1 is sender and h2 is receiver)
*****For throughput calculation we calculate the total length of sent packets from sender and divide it by the interval for which the iperf command run after which we convert bytes into K bits using appropriate conversion methods

*****For normalised throughput calculation we find the total sent packets by sender and total received packets by the receiver. So, normalised throughput = received packets/ sent packets.

*****For packet loss we find the sent packets from sender and see how many of those couldn’t reach the receiver which gives the number of packets loss and upon dividing it by the number of packets sent, we get the packet loss in percentage

Observation

1.The different types of packets in tcp packet capture are as follows:



The fields above for each packet are :
<Sender's ip> ">" <receiver's ip> <Flag> <sequence number> <acknowledgement number> <window size> <options> <length of packet in bytes>

The various types of packets observed are :
The three types of packets are *ip*, *utcp*, and *ctcp, which are printed first.*

There are 8 bits in the control bits section of the TCP header:
CWR | *ECE* | *URG* | *ACK* | *PSH* | *RST* | *SYN* | *FIN*
TCP uses a 3-way handshake protocol when it initializes a new connection; the connection sequence with regard to the TCP control bits is
1) Caller sends SYN
2) Recipient responds with SYN, ACK
3) Caller sends ACK

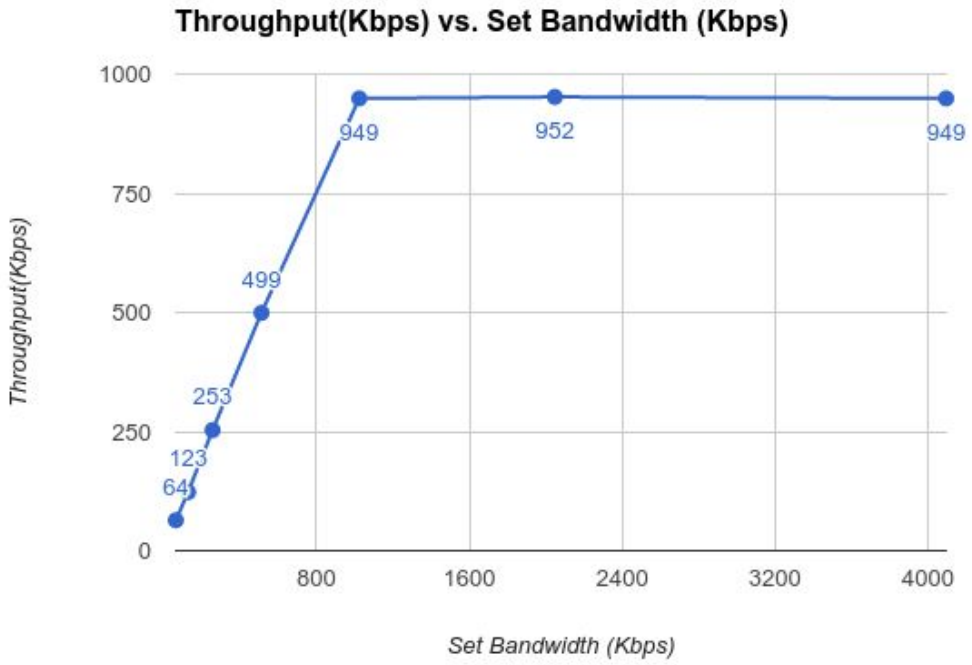
This we get to know by the flag
Flags are some combination of S (SYN), F (FIN), P (PUSH), R (RST), U (URG), W (ECN CWR), E (ECN-Echo) or `.' (ACK), or `none' if no flags are set

So here too we can see there are ACK, PUSH, FIN, SYN packets primarily and some packets which are combination of some of the above four flags like FIN-PUSH-ACK.

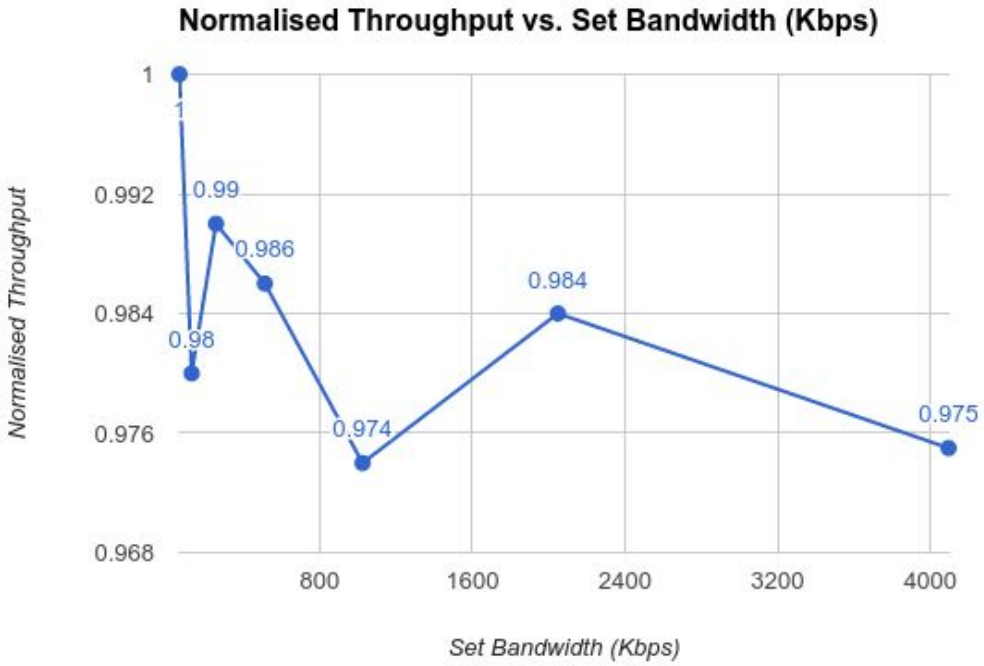
2. A)
The normalised TCP goodput is as follows :

Normalized TCP Goodput = total amount of data received / total amount of data sent
= 1571105/1571105
= 1

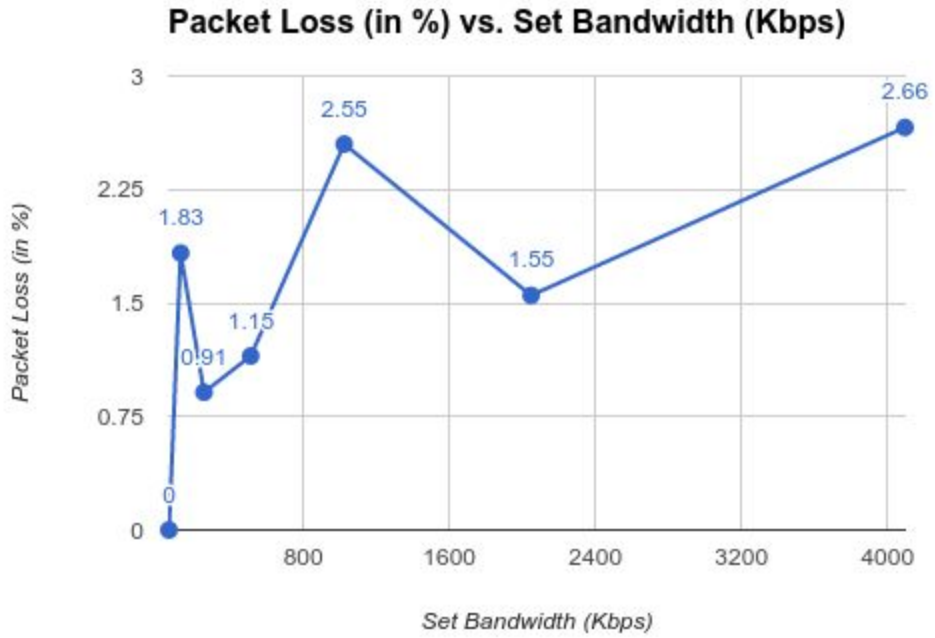
B)



C)


Normalised Throughput vs. Set Bandwidth (Kbps)
Normalised Throughput
1
0.992
0.984
0.976
0.968
1
0.99
0.98
0.986
0.974
0.984
0.975
800
1600
2400
3200
4000
Set Bandwidth (Kbps)

D)


Packet Loss (in %) vs. Set Bandwidth (Kbps)
Packet Loss (in %)
3
2.25
1.5
0.75
0
0
1.83
0.91
1.15
2.55
1.55
2.66
800
1600
2400
3200
4000
Set Bandwidth (Kbps)

CONCLUSION

From the graph in part B , we observe that throughput increases with bandwidth but then saturates at 1024 kbits/sec which is trivial as bandwidth of our topology is 1024 kbits/sec. For part C and part D we observe that normalised throughput decreases with bandwidth and packet loss increases with bandwidth. This is because UDP has no congestion control algorithm as compared to TCP so when bandwidth increases, number of packets transmitted increases which leads to increased congestion and therefore the number of packets getting dropped increases and data loss increases which leads to increased packet loss and decreased Normalised throughput.

Part 2

Procedure : We proceed as per part 1 modifying the first step that is forming the network topology.
We need to create custom topology for which we can use a python script.
In the python script, we create our desired topology using the mininet API and setting the bandwidth, loss and delay for the mentioned links. For the middle link that is the one between the switches, the configuration of bandwidth and delay is variable and hence will be changed to take throughput for other parts.

Now, copy the script in the home folder of mininet after ssh login and use the following to create the desired topology using the python script

```
>sudo mn --custom ~/mininet/topo-2sw-2host.py --topo mytopo --link tc
```

Now, we repeat part 1 for every subpart of part 2 that is for part a of part2, we repeat what we did in part 1 for various bandwidths of UDP and record the throughputs.

Now, after every part out of nine, we will update our middle link parameters in the python script kept in home directory of mininet, and form the topology again using the following two commands in succession :

```
>sudo mn -c
>sudo mn --custom ~/mininet/topo-2sw-2host.py --topo mytopo --link tc
```

Also to capture packets at s3 and s4, we type the following additional commands just before the iperf commands in the mininet CLI :

```
> s3 sudo tcpdump -i s3-eth1 -n -t > part2/udp_s3_eth1 &
> s3 sudo tcpdump -i s3-eth2 -n -t > part2/udp_s3_eth2 &
> s4 sudo tcpdump -i s4-eth1 -n -t > part2/udp_s4_eth1 &
> s4 sudo tcpdump -i s4-eth2 -n -t > part2/udp_s4_eth2 &
```

Here, eth1 and eth2 are the two interfaces of the switch s3 and s4.

(For UDP, h1 is sender and h2 is receiver)
*****For throughput calculation we calculate the total length of sent packets from sender and divide it by the interval for which the iperf command run after which we convert bytes into K bits using appropriate conversion methods

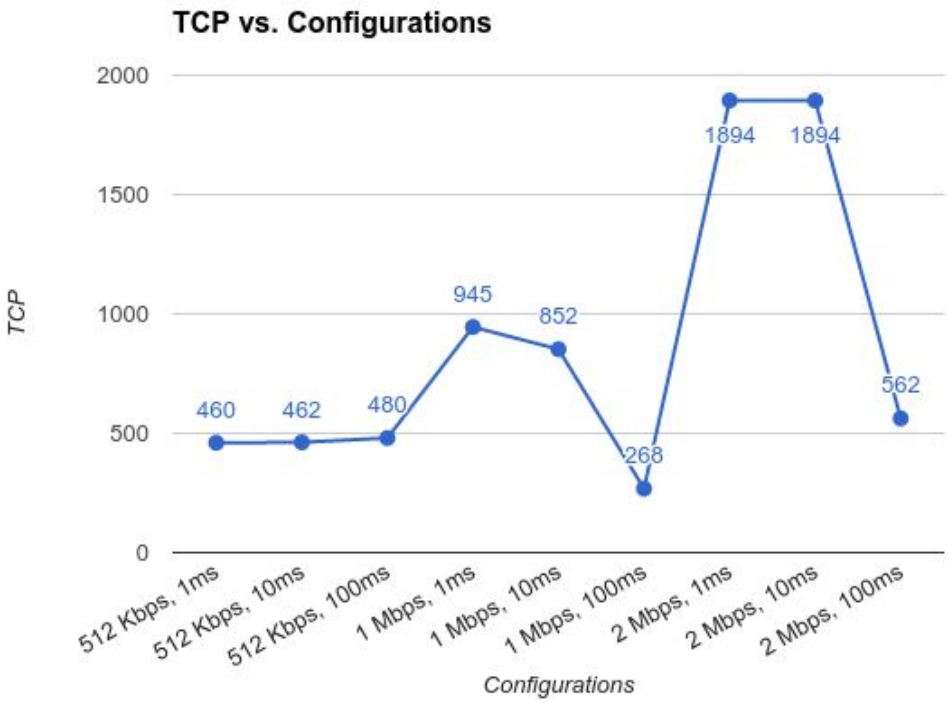
*****For normalised throughput calculation we find the total sent packets by sender and total received packets by the receiver. So, normalised throughput = received packets/ sent packets.

*****For packet loss we find the sent packets from sender and see how many of those couldn't reach the receiver which gives the number of packets loss and upon dividing it by the number of packets sent, we get the packet loss in percentage.

---For loss at S3, we find the difference in the number of sent packages from h1 that reach S3 and are forwarded from S3, this gives the loss in number of packets by S3 and dividing it by number of packets sent by h1, we get fraction of packet loss made by S3...Similarly, we repeat for S4 and h2 and calculate their respective loss fractions

Observation :

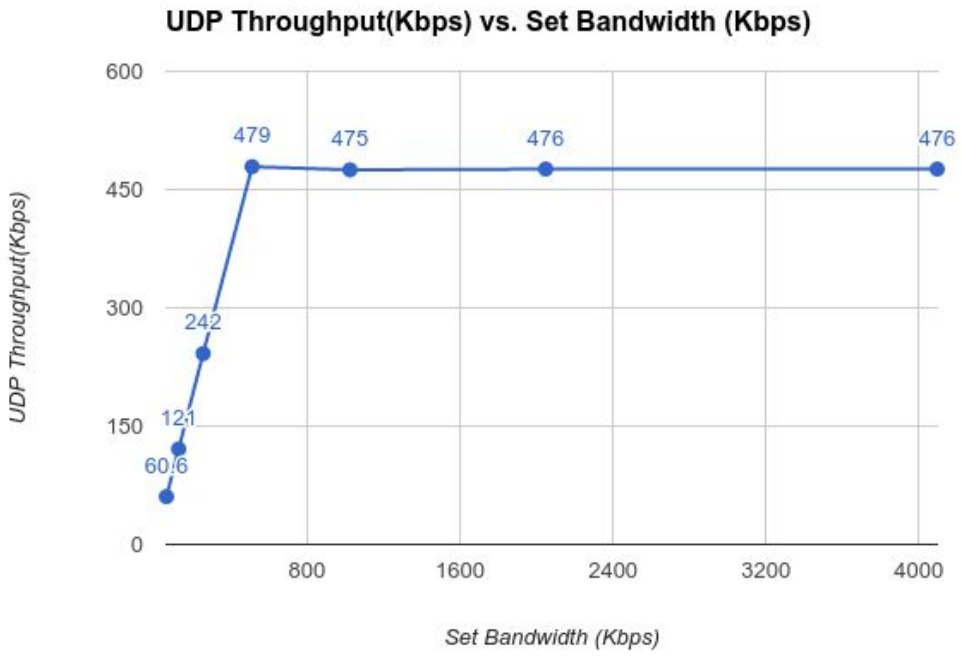
The TCP bandwidth observed for every of the nine configurations of the middle (between two switches) link :

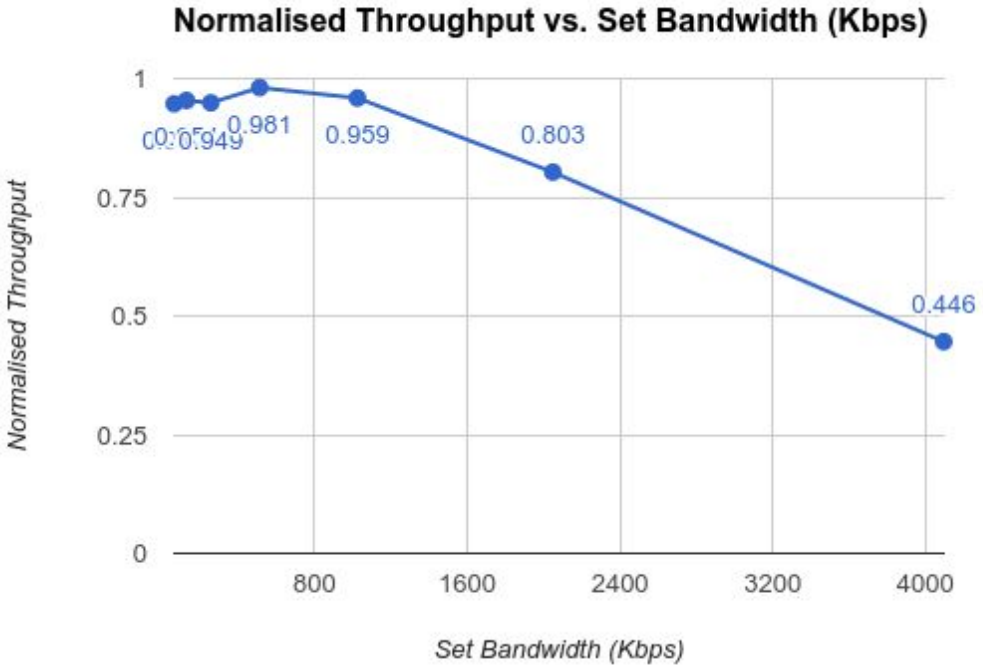


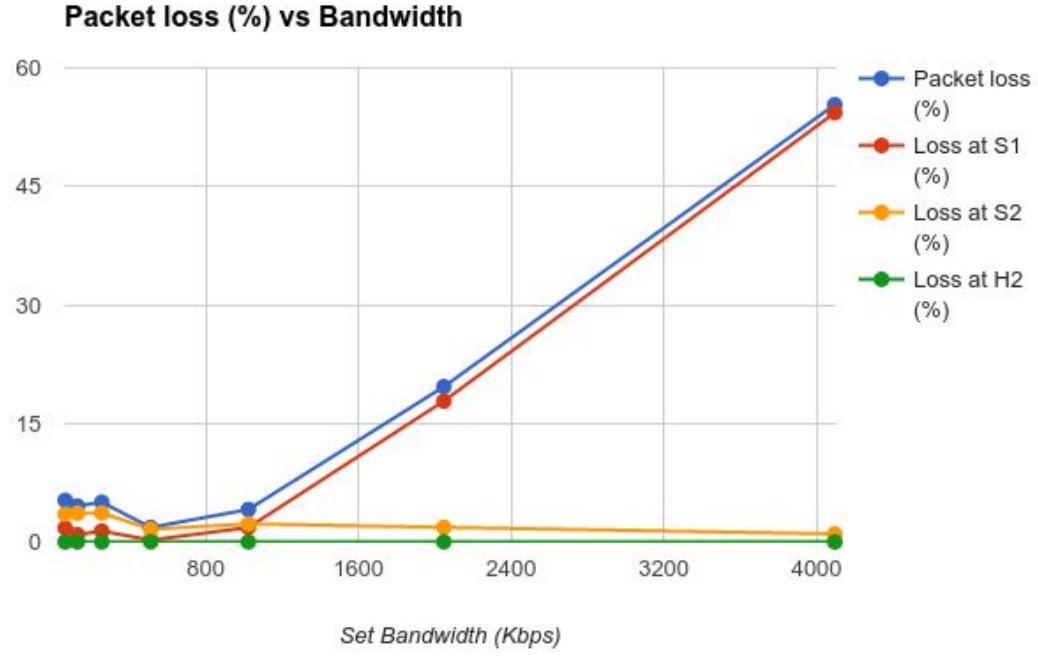
1.Bandwidth = 512 Kbps, Delay = 1 ms

The normalised TCP goodput is as follows :

Normalized TCP Goodput = total amount of data received / total amount of data sent
= 917529/917529
= 1



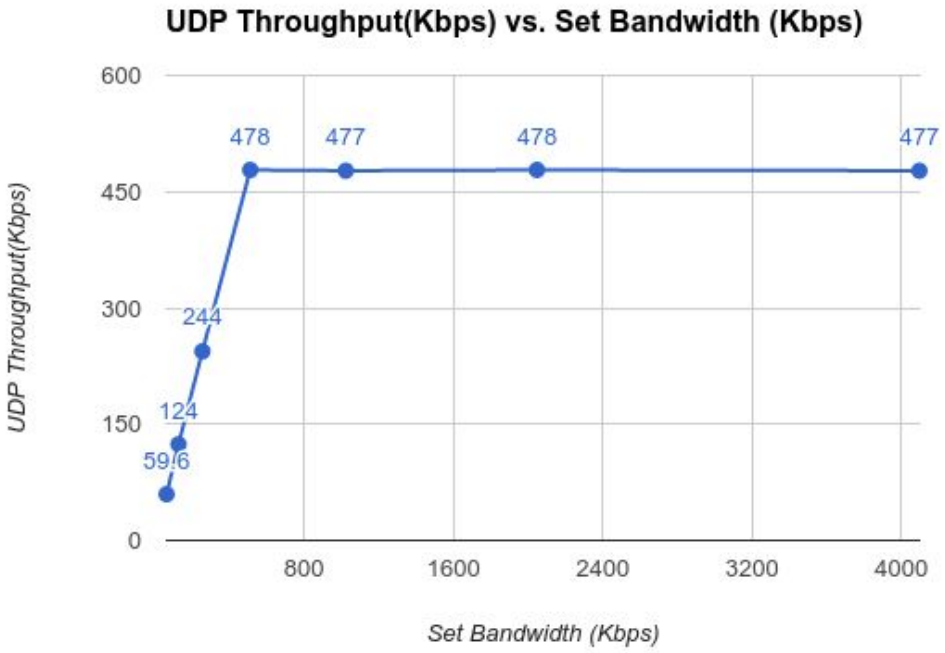

Normalised Throughput vs. Set Bandwidth (Kbps)
Normalised Throughput
1
0.75
0.5
0.25
0
0.949
0.981
0.959
0.803
0.446
800
1600
2400
3200
4000
Set Bandwidth (Kbps)

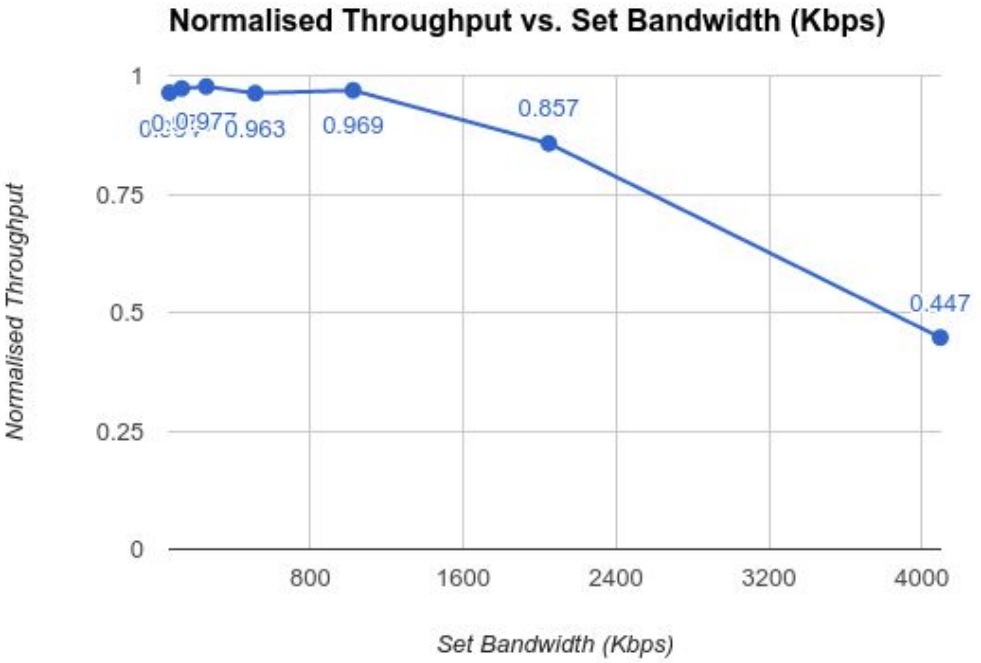

Packet loss (%) vs Bandwidth
60
45
30
15
0
800
1600
2400
3200
4000
Set Bandwidth (Kbps)
Packet loss (%)
Loss at S1 (%)
Loss at S2 (%)
Loss at H2 (%)

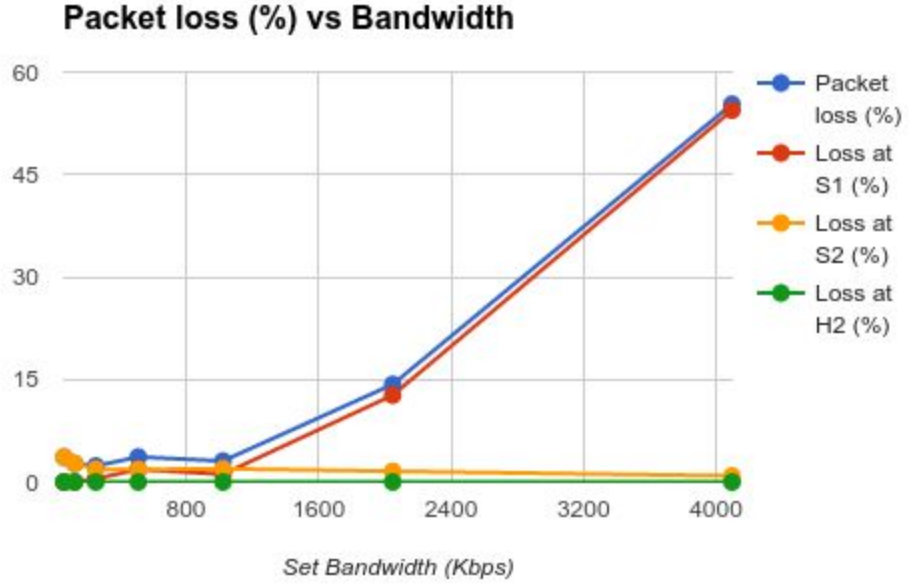
2.Bandwidth = 512 Kbps, Delay = 10 ms

The normalised TCP goodput is as follows :

Normalized TCP Goodput = total amount of data received / total amount of data sent
= 786457/786457
= 1



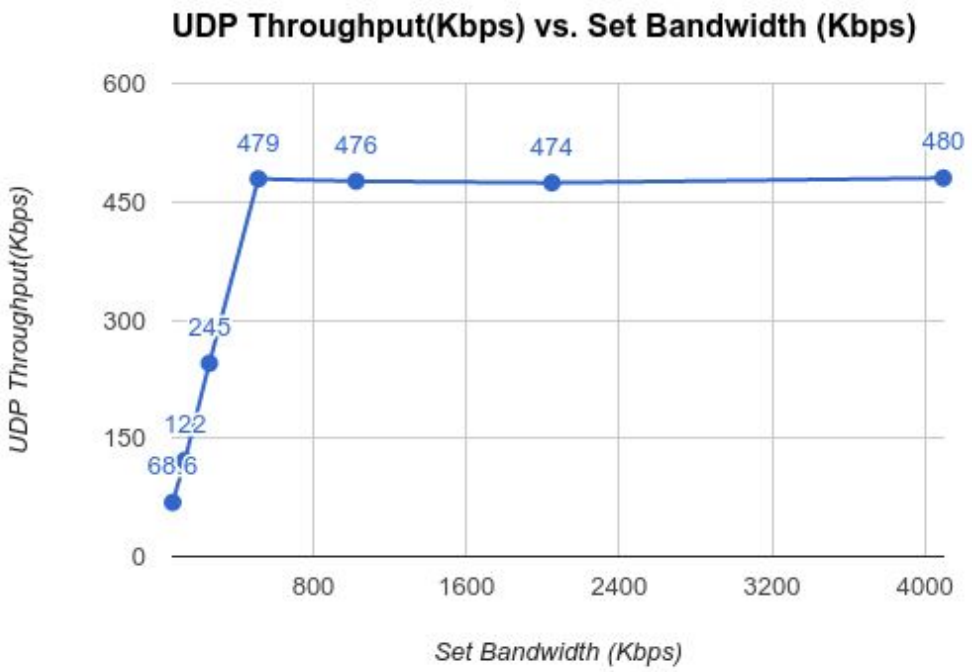

Normalised Throughput vs. Set Bandwidth (Kbps)
Normalised Throughput
1
0.75
0.5
0.25
0
0.963
0.969
0.857
0.447
800
1600
2400
3200
4000
Set Bandwidth (Kbps)

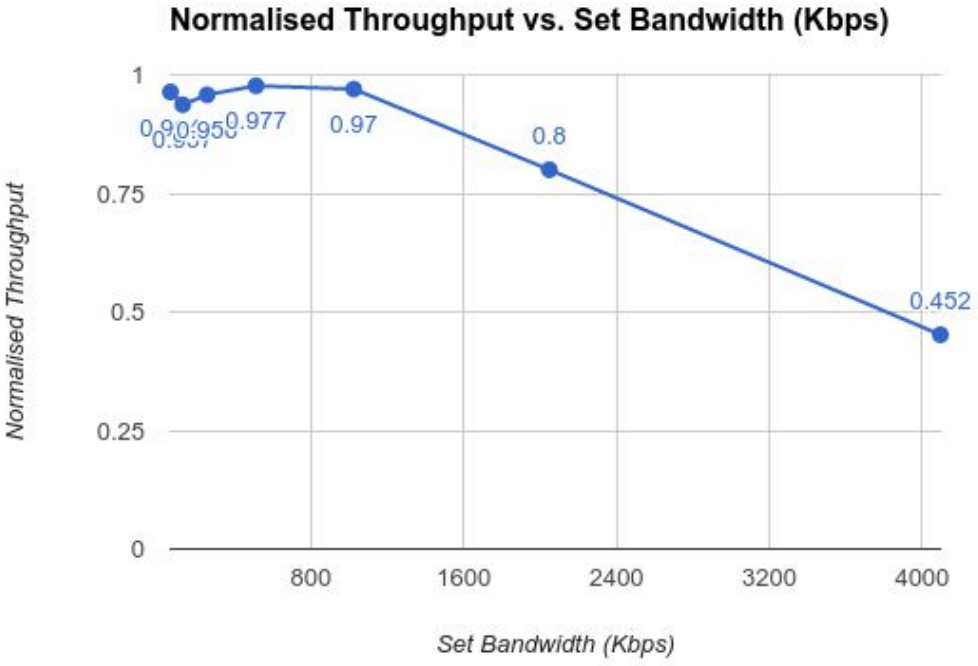

Packet loss (%) vs Bandwidth
60
45
30
15
0
800
1600
2400
3200
4000
Set Bandwidth (Kbps)
Packet loss (%)
Loss at S1 (%)
Loss at S2 (%)
Loss at H2 (%)

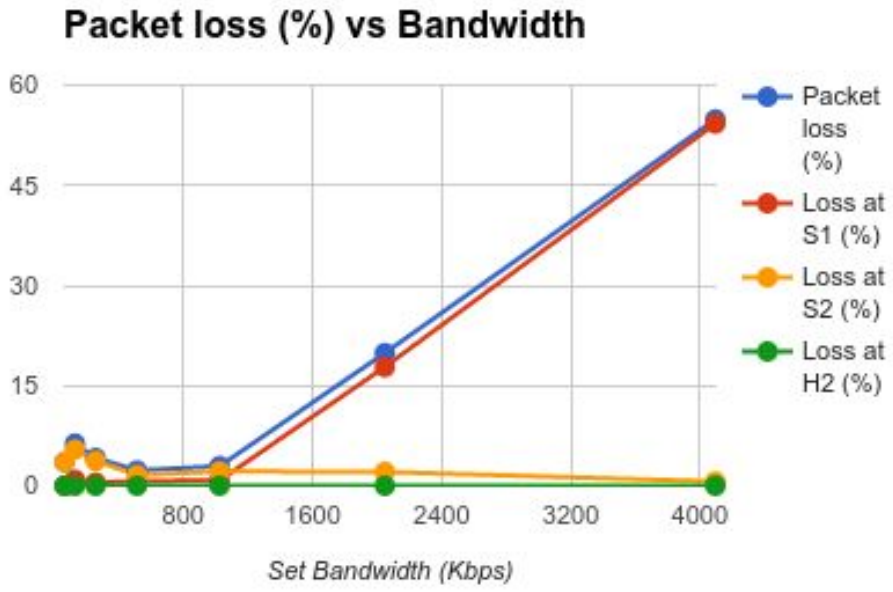
3.Bandwidth = 512 Kbps, Delay = 100 ms

The normalised TCP goodput is as follows :

Normalized TCP Goodput = total amount of data received / total amount of data sent
= 907921/907921
= 1



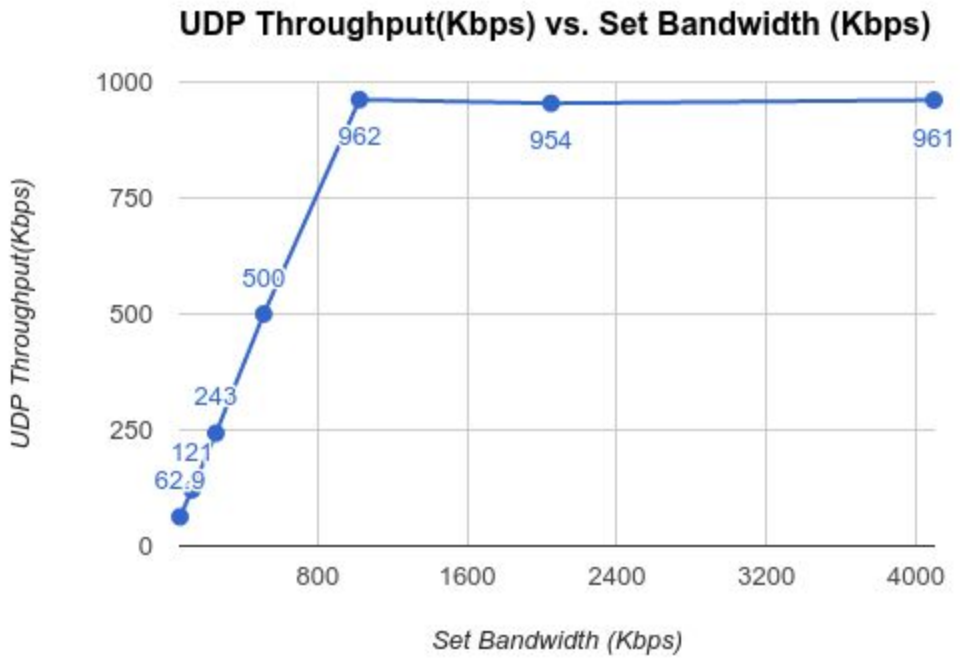

Normalised Throughput vs. Set Bandwidth (Kbps)
Normalised Throughput
1
0.75
0.5
0.25
0
0.977
0.97
0.8
0.452
800
1600
2400
3200
4000
Set Bandwidth (Kbps)

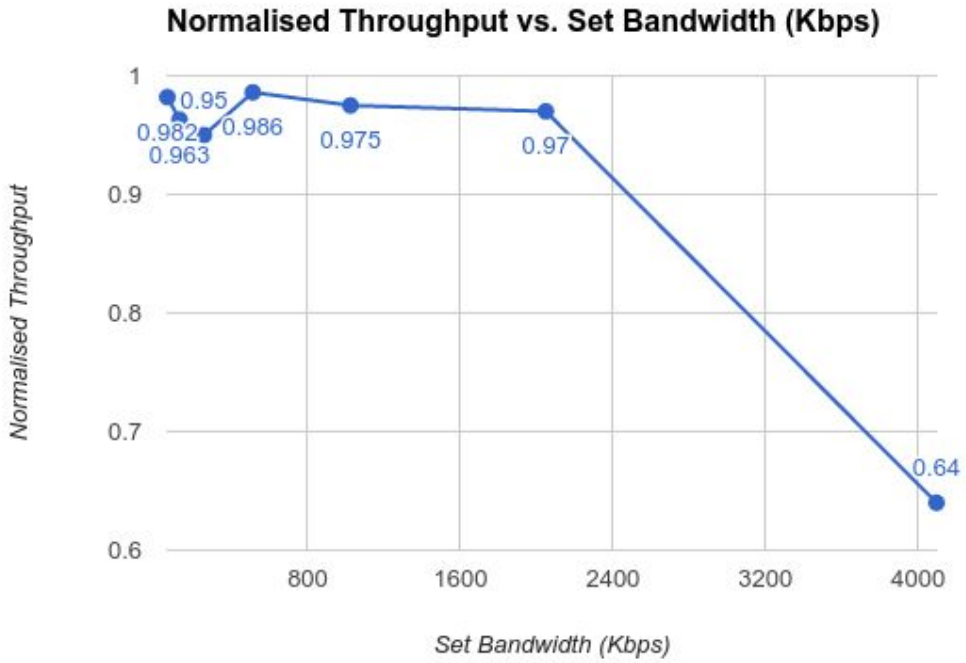

Packet loss (%) vs Bandwidth
60
45
30
15
0
800
1600
2400
3200
4000
Set Bandwidth (Kbps)
Packet loss (%)
Loss at S1 (%)
Loss at S2 (%)
Loss at H2 (%)

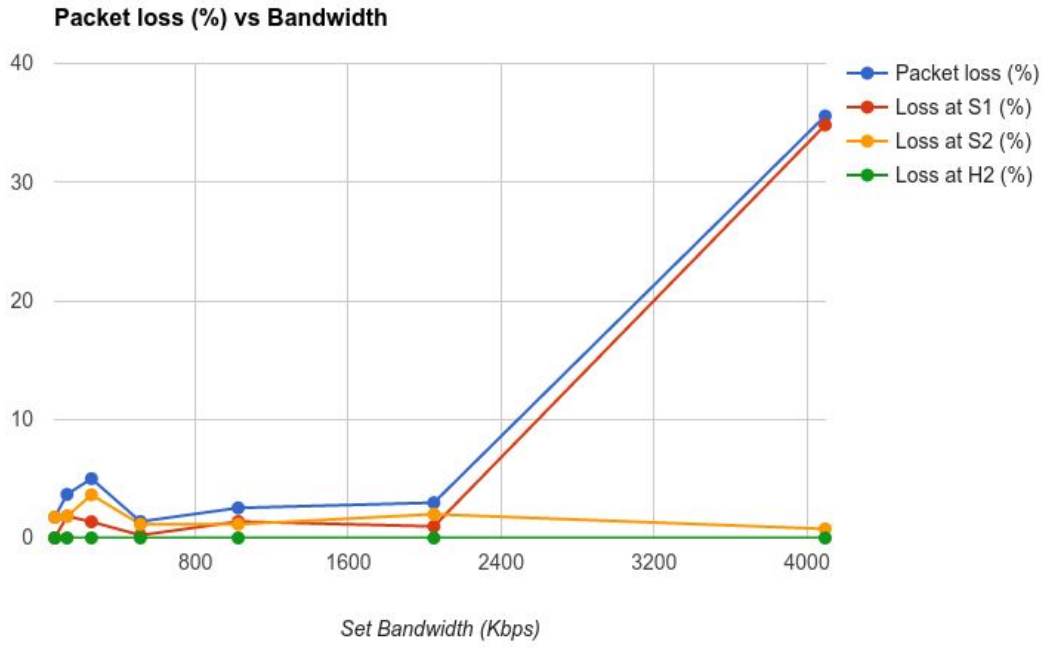
4.Bandwidth = 512 Kbps, Delay = 1 ms

The normalised TCP goodput is as follows :

Normalized TCP Goodput = total amount of data received / total amount of data sent
= 1441817/1441817
= 1



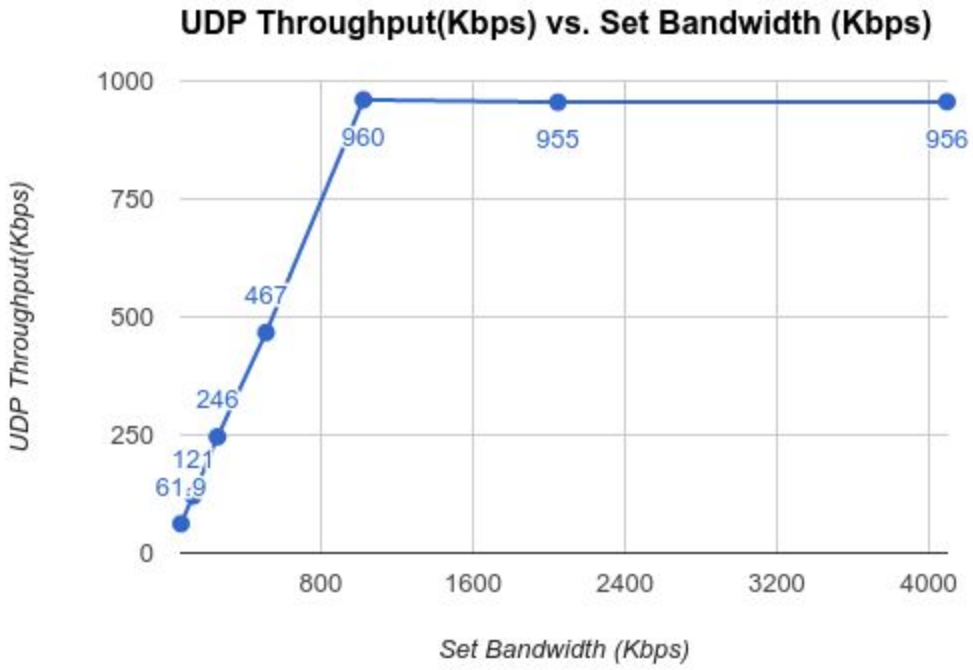

Normalised Throughput vs. Set Bandwidth (Kbps)
Normalised Throughput
1
0.9
0.8
0.7
0.6
0.95
0.982
0.963
0.986
0.975
0.97
0.64
800
1600
2400
3200
4000
Set Bandwidth (Kbps)

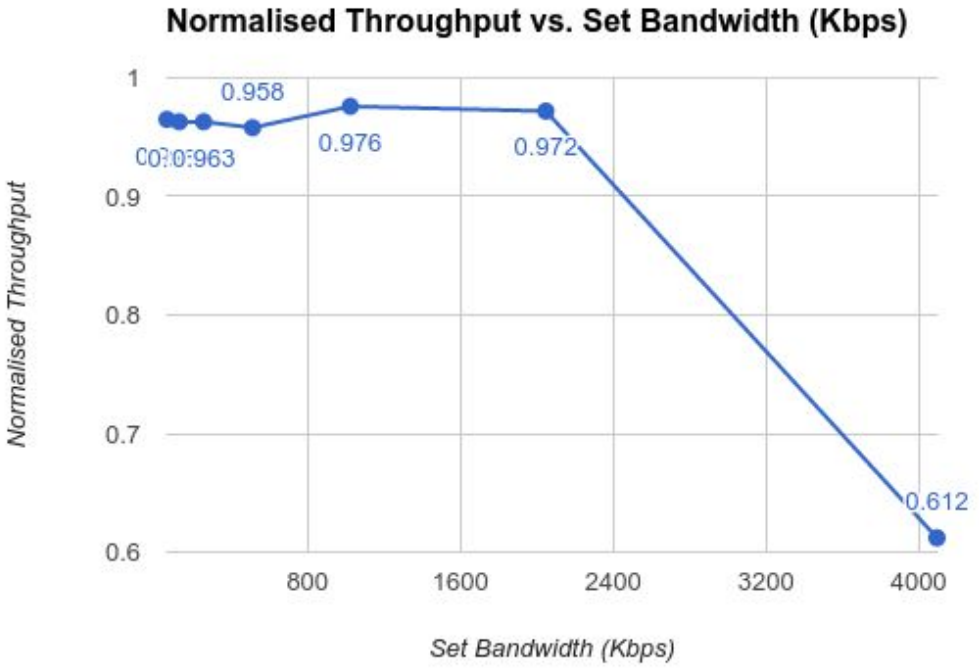

Packet loss (%) vs Bandwidth
40
30
20
10
0
800
1600
2400
3200
4000
Set Bandwidth (Kbps)
Packet loss (%)
Loss at S1 (%)
Loss at S2 (%)
Loss at H2 (%)

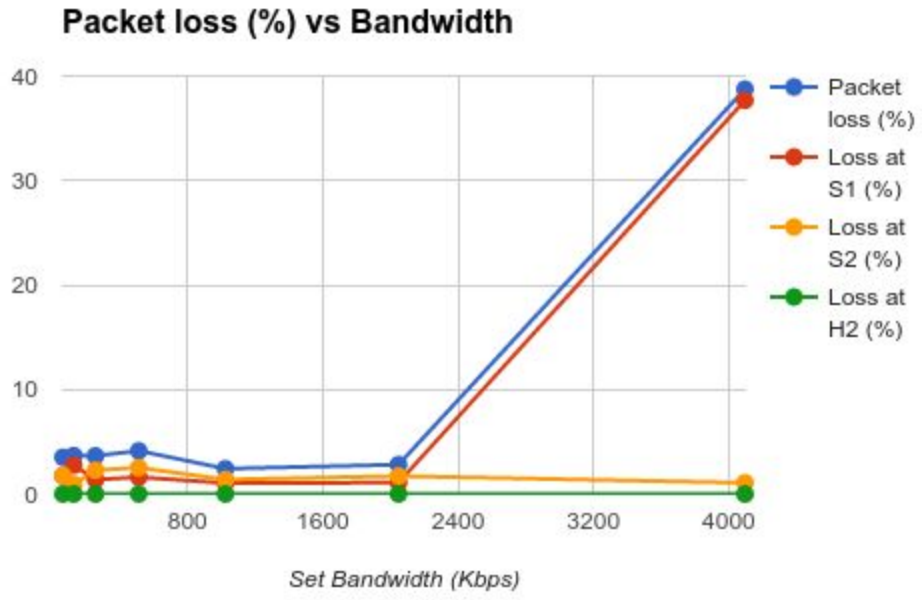
5.Bandwidth = 1 Mbps, Delay = 10 ms

The normalised TCP goodput is as follows :

Normalized TCP Goodput = total amount of data received / total amount of data sent
= 1310745/1310745
= 1



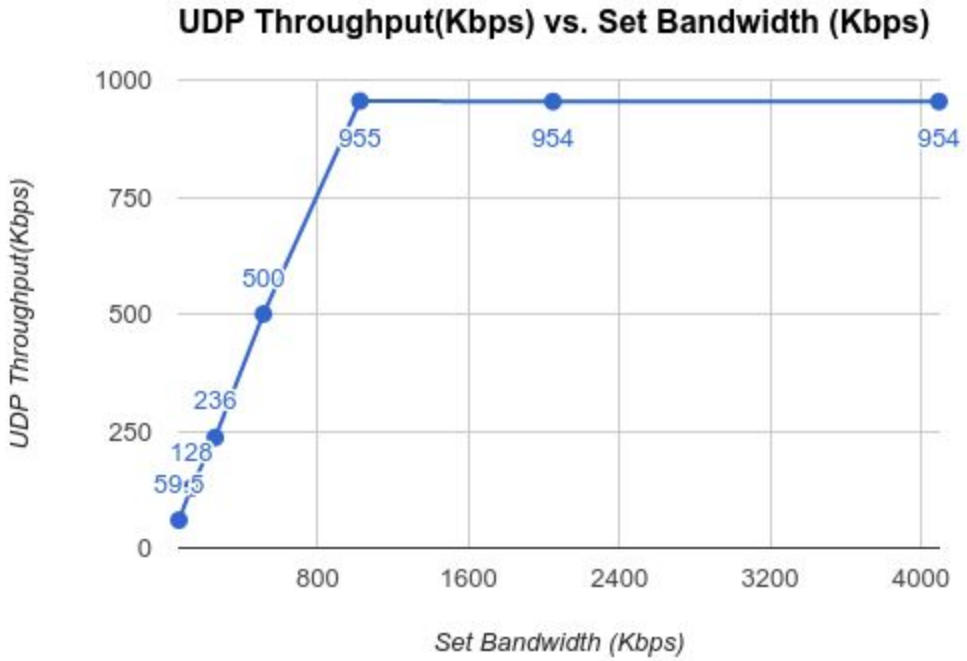


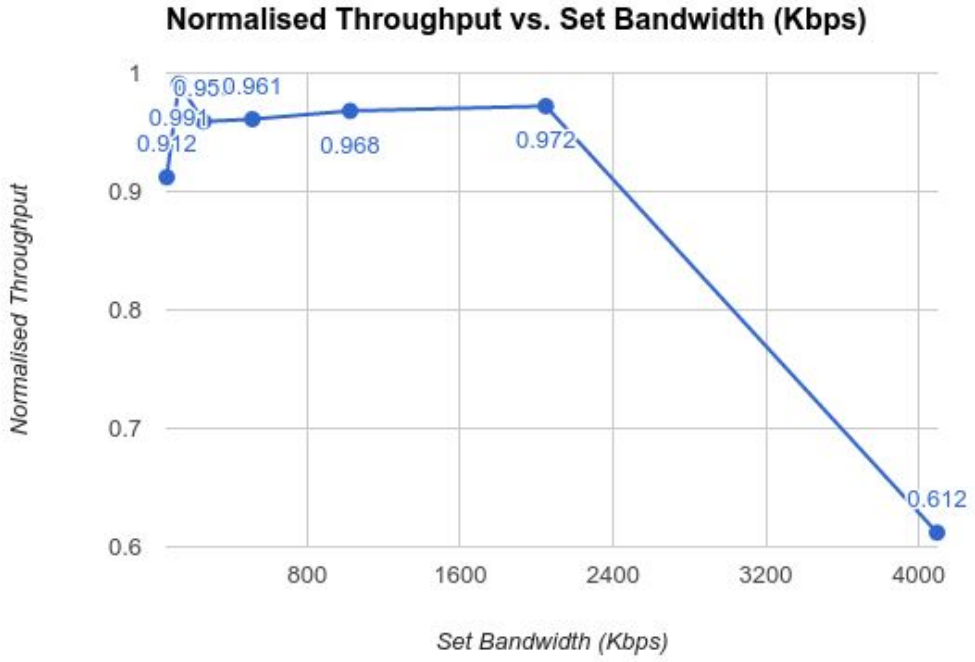


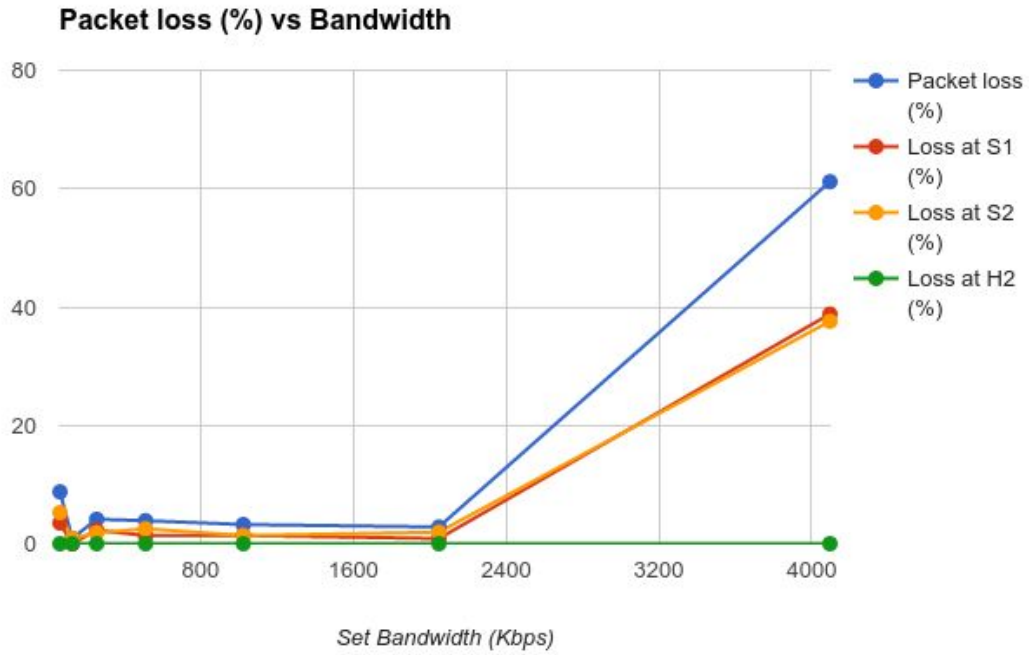
6.Bandwidth = 1 Mbps, Delay = 100 ms

The normalised TCP goodput is as follows :

Normalized TCP Goodput = total amount of data received / total amount of data sent
= 786289/786289
= 1



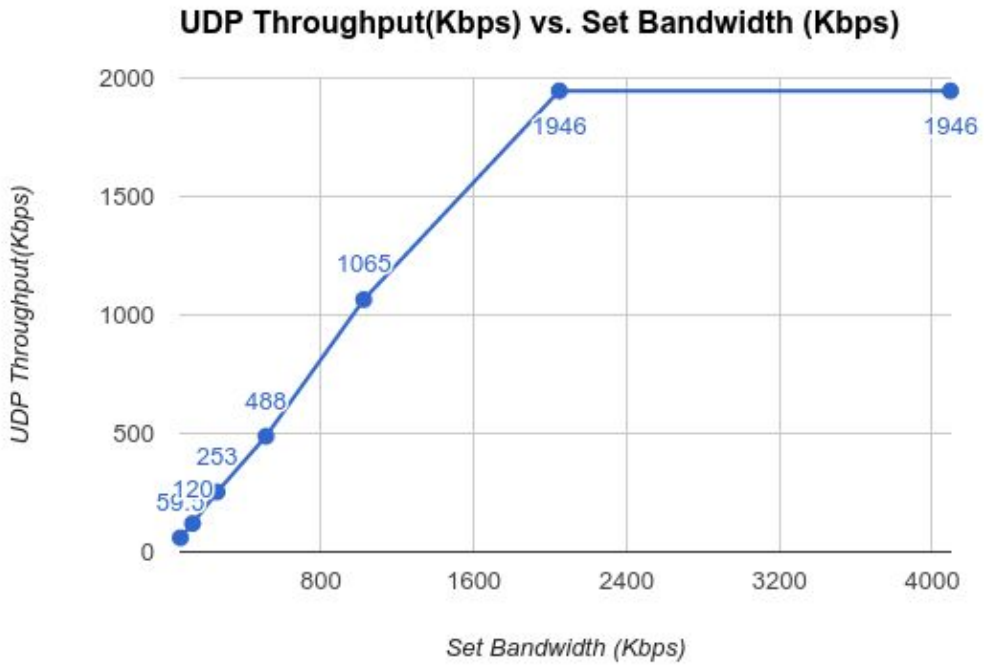

Normalised Throughput vs. Set Bandwidth (Kbps)
Normalised Throughput
1
0.9
0.8
0.7
0.6
0.95
0.961
0.991
0.912
0.968
0.972
0.612
800
1600
2400
3200
4000
Set Bandwidth (Kbps)

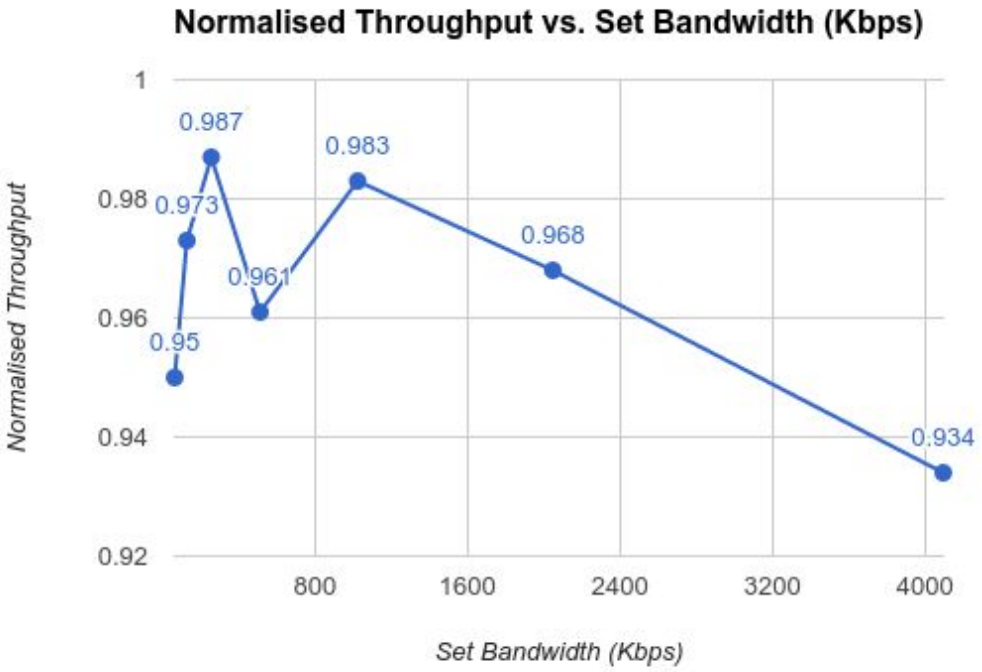

Packet loss (%) vs Bandwidth
80
60
40
20
0
800
1600
2400
3200
4000
Set Bandwidth (Kbps)
Packet loss (%)
Loss at S1 (%)
Loss at S2 (%)
Loss at H2 (%)

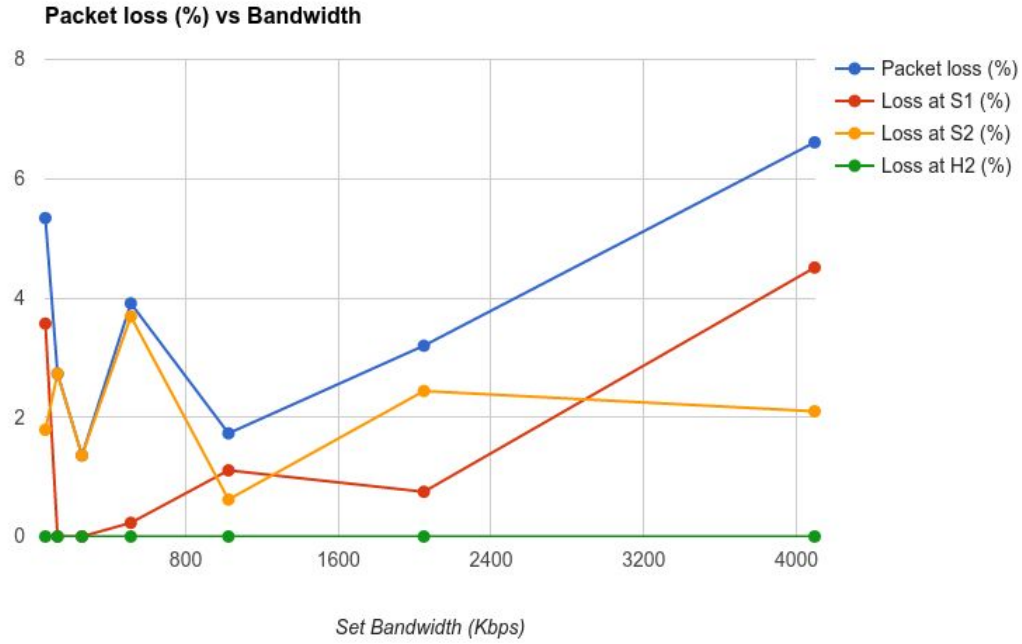
7.Bandwidth = 2 Mbps, Delay = 1 ms

The normalised TCP goodput is as follows :

Normalized TCP Goodput = total amount of data received / total amount of data sent
= 2621465/2621465
= 1



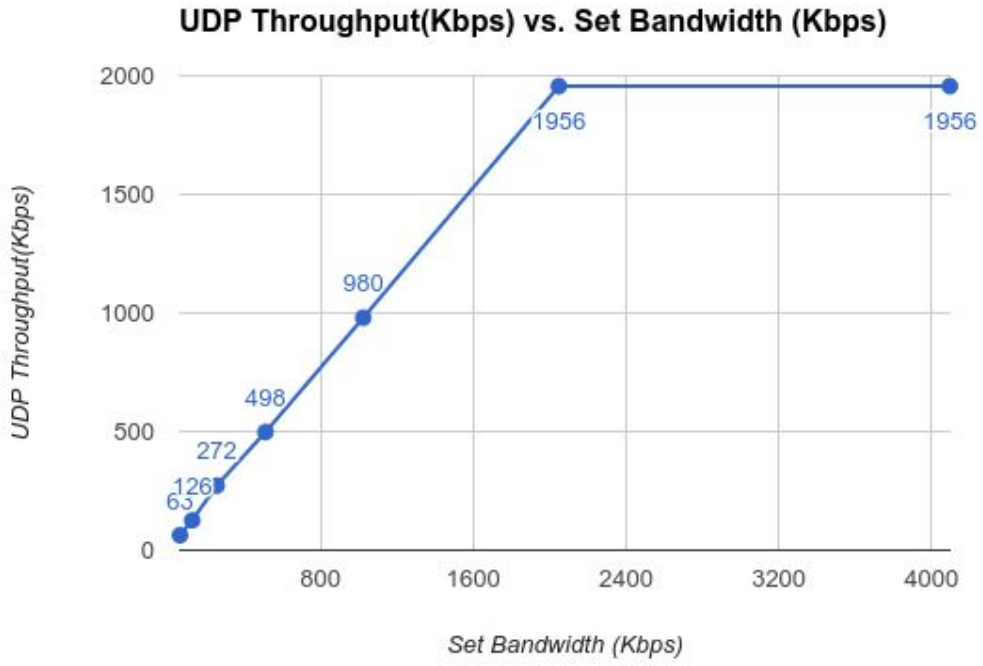

Normalised Throughput vs. Set Bandwidth (Kbps)
Normalised Throughput
1
0.98
0.96
0.94
0.92
0.987
0.983
0.973
0.968
0.961
0.95
0.934
800
1600
2400
3200
4000
Set Bandwidth (Kbps)

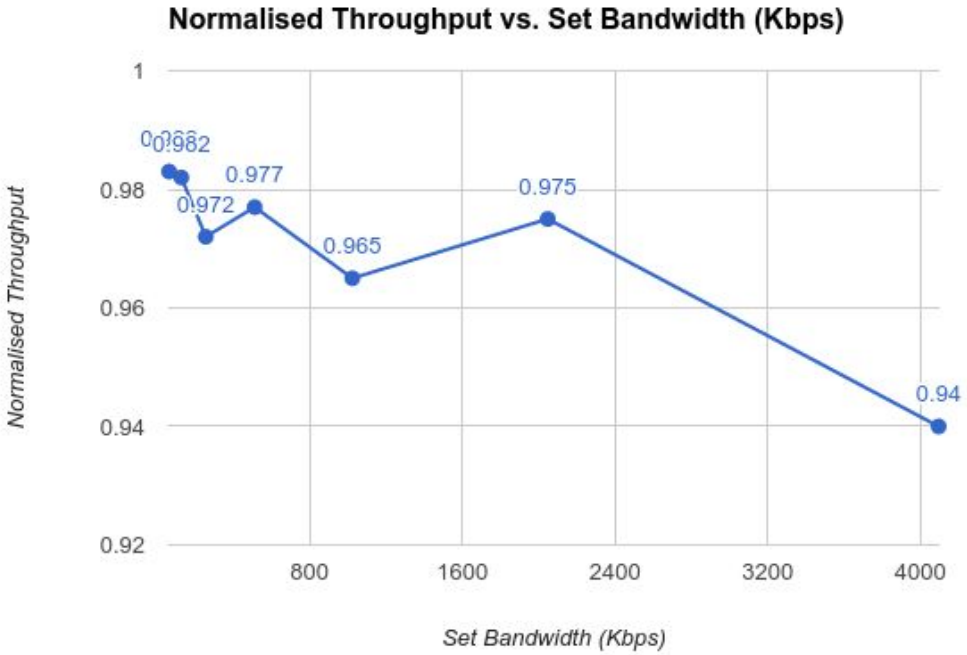

Packet loss (%) vs Bandwidth
8
6
4
2
0
800
1600
2400
3200
4000
Set Bandwidth (Kbps)
Packet loss (%)
Loss at S1 (%)
Loss at S2 (%)
Loss at H2 (%)

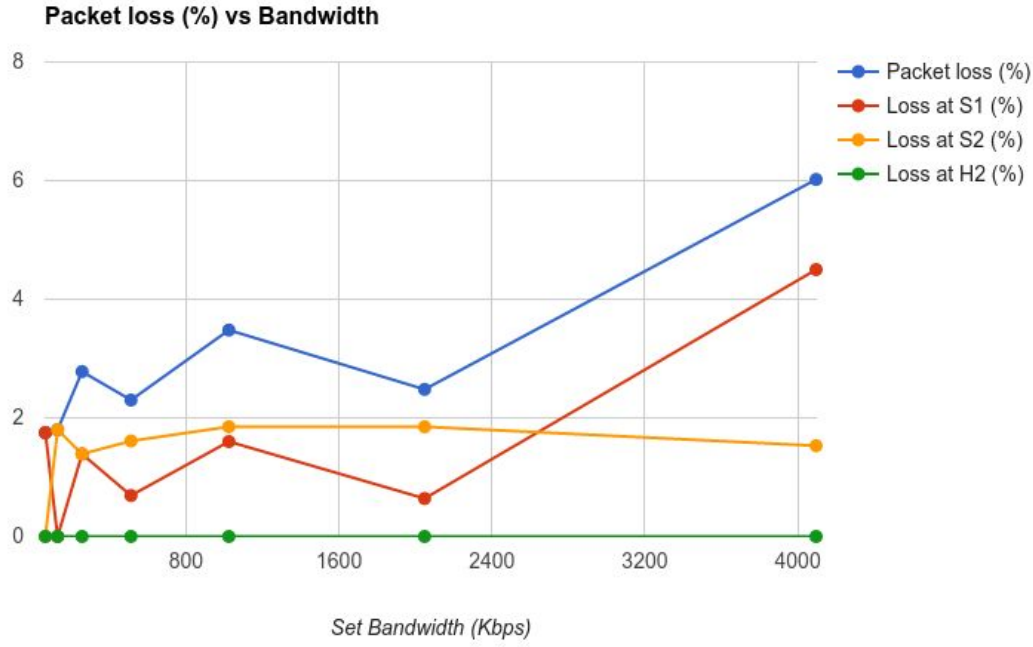
8.Bandwidth = 2 Mbps, Delay = 10 ms

The normalised TCP goodput is as follows :

Normalized TCP Goodput = total amount of data received / total amount of data sent
= 2751225/2751225
= 1



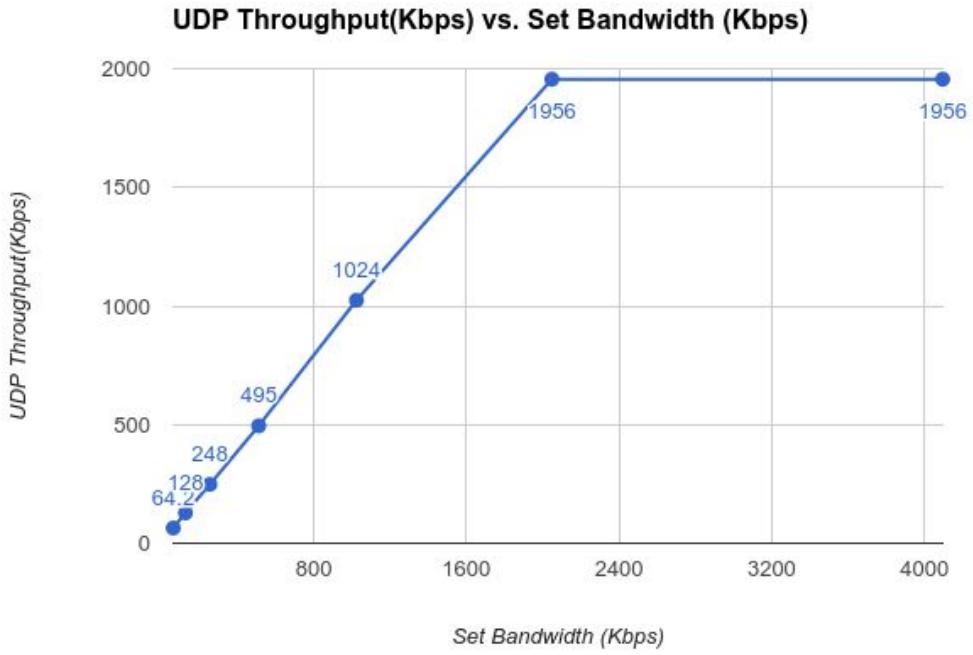

Normalised Throughput vs. Set Bandwidth (Kbps)
Normalised Throughput
1
0.98
0.96
0.94
0.92
0.982
0.977
0.972
0.965
0.975
0.94
800
1600
2400
3200
4000
Set Bandwidth (Kbps)

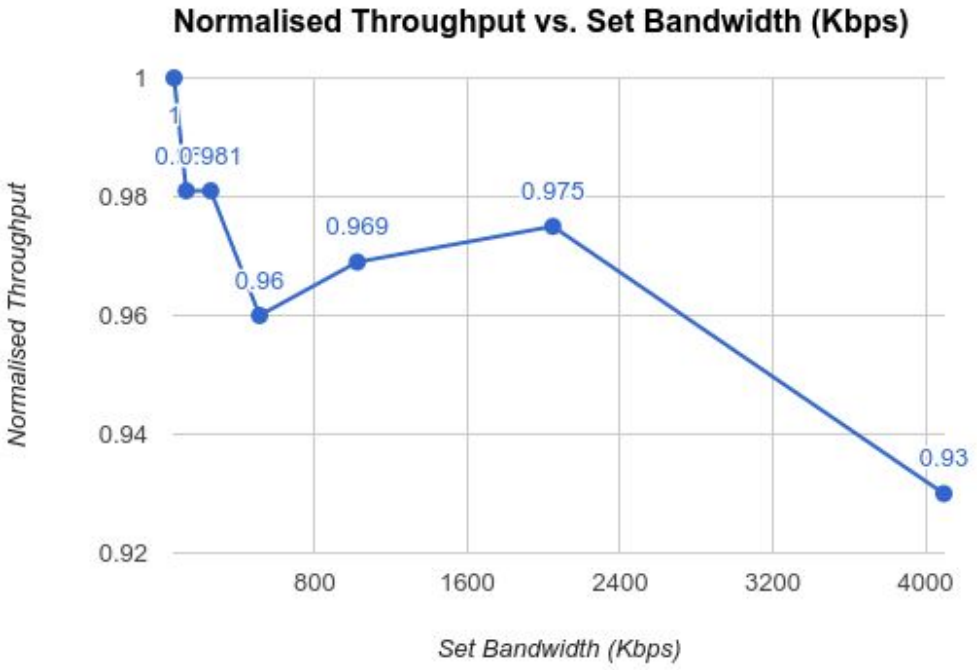

Packet loss (%) vs Bandwidth
8
6
4
2
0
800
1600
2400
3200
4000
Set Bandwidth (Kbps)
Packet loss (%)
Loss at S1 (%)
Loss at S2 (%)
Loss at H2 (%)

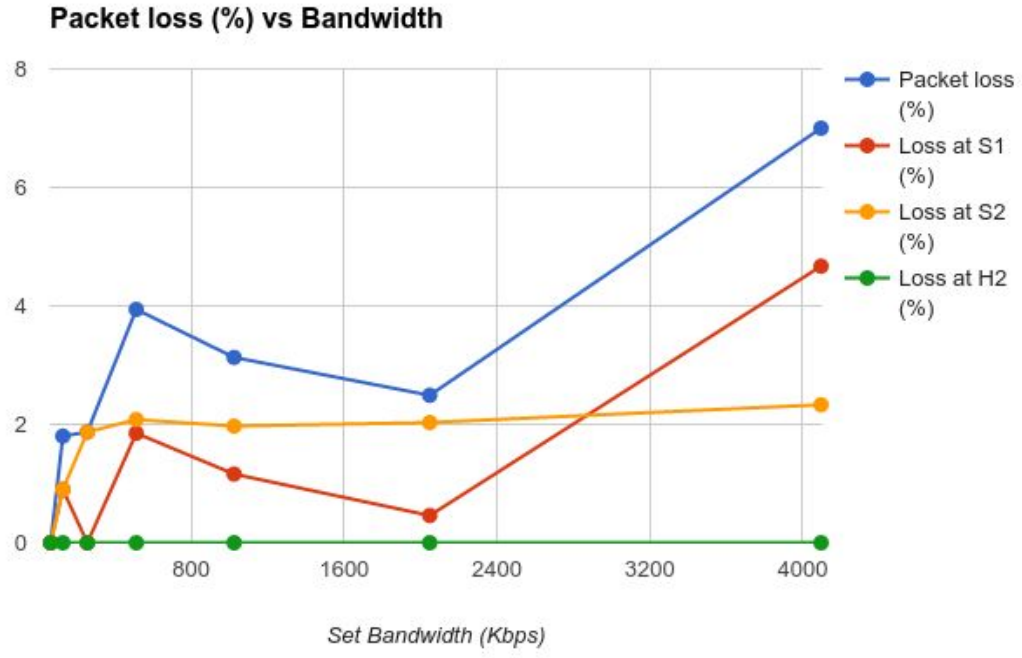
9.Bandwidth = 2 Mbps, Delay = 100 ms

The normalised TCP goodput is as follows :

Normalized TCP Goodput = total amount of data received / total amount of data sent
= 1048377/1048377
= 1




Normalised Throughput vs. Set Bandwidth (Kbps)
Normalised Throughput
1
0.98
0.96
0.94
0.92
1
0.0981
0.96
0.969
0.975
0.93
800
1600
2400
3200
4000
Set Bandwidth (Kbps)


Packet loss (%) vs Bandwidth
8
6
4
2
0
800
1600
2400
3200
4000
Set Bandwidth (Kbps)
Packet loss (%)
Loss at S1 (%)
Loss at S2 (%)
Loss at H2 (%)

CONCLUSION

In all the above cases we see that UDP throughput increases with bandwidth until it saturates which is the set bandwidth of our configuration. Comparing UDP Normalised throughput and packet loss for various bandwidths we observe that as bandwidth increases the normalised throughput decreases and packet loss increases as compared to previous set bandwidth because UDP has no congestion control algorithm as compared to TCP so when bandwidth increases, number of packets transmitted increases which leads to increased congestion and therefore the number of packets getting dropped increases and data loss increases which leads to increased packet loss and decreased Normalised throughput.

Another very interesting feature is maximum packet loss occurs at S1 itself mostly when the bandwidth of the middle link is less than 1 Mbps and no packet loss at H2. This is probably because only way for packets to get lost at H2 is when there is loss in the link connecting S2 to H2 and since this is a virtual simulator of networks, there is no channel loss and hence no loss is observed at H2.
When the middle link's bandwidth is less than 1 Mbps there is a noticeable difference in the throughput observed incoming at S1 and output bandwidth allowable from the output of S1, since the incoming S1's link's bandwidth is 3Mbps. So at S1 for initial cases, input data is appreciably more than the maximum allowable output bandwidth leading to most of the packet loss at S1 itself where total packet loss is around 50 % and S1's contribution accounts for about 98 % of the packet loss. Increased packet loss leads to decreased normalised throughput.

Authored by **:** **14CS10006 (Apurv Kumar)**
14CS30034 (Shubham Sharma)