

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

>lperf -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
```

Now,we get the required TCP throughput reading from the bandwidth of the TCP server that 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 :


```

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:

```
IP 10.0.0.2.5001 > 10.0.0.1.56524: Flags [.], ack 554609, win 838, options [nop,nop,TS val 10942315 ecr 10942297], length 0
IP 10.0.0.1.56524 > 10.0.0.2.5001: Flags [.], seq 554609:557505, ack 1, win 58, options [nop,nop,TS val 10942303 ecr 10942302], length 2896
IP 10.0.0.2.5001 > 10.0.0.1.56524: Flags [.], ack 557505, win 838, options [nop,nop,TS val 10942321 ecr 10942303], length 0
IP 10.0.0.1.56524 > 10.0.0.2.5001: Flags [.], seq 557505:560401, ack 1, win 58, options [nop,nop,TS val 10942309 ecr 10942309], length 2896
IP 10.0.0.2.5001 > 10.0.0.1.56524: Flags [.], ack 560401, win 838, options [nop,nop,TS val 10942327 ecr 10942309], length 0
IP 10.0.0.1.56524 > 10.0.0.2.5001: Flags [.], seq 560401:563297, ack 1, win 58, options [nop,nop,TS val 10942315 ecr 10942315], length 2896
IP 10.0.0.2.5001 > 10.0.0.1.56524: Flags [.], ack 563297, win 838, options [nop,nop,TS val 10942333 ecr 10942315], length 0
IP 10.0.0.1.56524 > 10.0.0.2.5001: Flags [.], seq 563297:566193, ack 1, win 58, options [nop,nop,TS val 10942321 ecr 10942321], length 2896
IP 10.0.0.2.5001 > 10.0.0.1.56524: Flags [.], ack 566193, win 838, options [nop,nop,TS val 10942339 ecr 10942321], length 0
IP 10.0.0.1.56524 > 10.0.0.2.5001: Flags [P.], seq 566193:569089, ack 1, win 58, options [nop,nop,TS val 10942327 ecr 10942327], length 2896
IP 10.0.0.2.5001 > 10.0.0.1.56524: Flags [.], ack 569089, win 838, options [nop,nop,TS val 10942345 ecr 10942327], length 0
IP 10.0.0.1.56524 > 10.0.0.2.5001: Flags [.], seq 569089:571985, ack 1, win 58, options [nop,nop,TS val 10942333 ecr 10942333], length 2896
IP 10.0.0.2.5001 > 10.0.0.1.56524: Flags [.], ack 571985, win 838, options [nop,nop,TS val 10942351 ecr 10942333], length 0
IP 10.0.0.1.56524 > 10.0.0.2.5001: Flags [.], seq 571985:574881, ack 1, win 58, options [nop,nop,TS val 10942339 ecr 10942339], length 2896
IP 10.0.0.2.5001 > 10.0.0.1.56524: Flags [.], ack 574881, win 838, options [nop,nop,TS val 10942357 ecr 10942339], length 0
IP 10.0.0.1.56524 > 10.0.0.2.5001: Flags [.], seq 574881:577777, ack 1, win 58, options [nop,nop,TS val 10942345 ecr 10942345], length 2896
IP 10.0.0.2.5001 > 10.0.0.1.56524: Flags [.], ack 577777, win 838, options [nop,nop,TS val 10942363 ecr 10942345], length 0
IP 10.0.0.1.56524 > 10.0.0.2.5001: Flags [.], seq 577777:580673, ack 1, win 58, options [nop,nop,TS val 10942351 ecr 10942351], length 2896
IP 10.0.0.2.5001 > 10.0.0.1.56524: Flags [.], ack 580673, win 838, options [nop,nop,TS val 10942369 ecr 10942351], length 0
IP 10.0.0.1.56524 > 10.0.0.2.5001: Flags [.], seq 583569:586465, ack 1, win 58, options [nop,nop,TS val 10942364 ecr 10942363], length 2896
IP 10.0.0.2.5001 > 10.0.0.1.56524: Flags [.], ack 580673, win 838, options [nop,nop,TS val 10942375 ecr 10942351,nop,nop,sack 1
{583569:586465}], length 0
ARP, Request who-has 10.0.0.1 tell 10.0.0.2, length 28
IP 10.0.0.1.56524 > 10.0.0.2.5001: Flags [.], seq 586465:589361, ack 1, win 58, options [nop,nop,TS val 10942364 ecr 10942363], length 2896
IP 10.0.0.2.5001 > 10.0.0.1.56524: Flags [.], ack 580673, win 838, options [nop,nop,TS val 10942381 ecr 10942351,nop,nop,sack 1
{583569:589361}], length 0
IP 10.0.0.1.56524 > 10.0.0.2.5001: Flags [.], seq 589361:592257, ack 1, win 58, options [nop,nop,TS val 10942370 ecr 10942369], length 2896
IP 10.0.0.2.5001 > 10.0.0.1.56524: Flags [.], ack 580673, win 838, options [nop,nop,TS val 10942387 ecr 10942351,nop,nop,sack 1
{583569:592257}], length 0
ARP, Reply 10.0.0.1 is-at da:a0:4b:dc:5b, length 28
IP 10.0.0.1.56524 > 10.0.0.2.5001: Flags [.], seq 580673:582121, ack 1, win 58, options [nop,nop,TS val 10942382 ecr 10942381], length 1448
IP 10.0.0.2.5001 > 10.0.0.1.56524: Flags [.], ack 582121, win 838, options [nop,nop,TS val 10942393 ecr 10942382,nop,nop,sack 1
{583569:592257}], length 0
IP 10.0.0.1.56524 > 10.0.0.2.5001: Flags [.], seq 582121:583569, ack 1, win 58, options [nop,nop,TS val 10942382 ecr 10942381], length 1448
IP 10.0.0.2.5001 > 10.0.0.1.56524: Flags [.], ack 592257, win 838, options [nop,nop,TS val 10942396 ecr 10942382], length 0
IP 10.0.0.1.56524 > 10.0.0.2.5001: Flags [.], seq 592257:593705, ack 1, win 58, options [nop,nop,TS val 10942382 ecr 10942381], length 1448
IP 10.0.0.2.5001 > 10.0.0.1.56524: Flags [.], ack 593705, win 838, options [nop,nop,TS val 10942399 ecr 10942382], length 0
IP 10.0.0.1.56524 > 10.0.0.2.5001: Flags [.], seq 593705:595153, ack 1, win 58, options [nop,nop,TS val 10942388 ecr 10942387], length 1448
IP 10.0.0.2.5001 > 10.0.0.1.56524: Flags [.], ack 595153, win 838, options [nop,nop,TS val 10942402 ecr 10942388], length 0
IP 10.0.0.1.56524 > 10.0.0.2.5001: Flags [.], seq 595153:596601, ack 1, win 58, options [nop,nop,TS val 10942388 ecr 10942387], length 1448
IP 10.0.0.2.5001 > 10.0.0.1.56524: Flags [.], ack 596601, win 838, options [nop,nop,TS val 10942406 ecr 10942388], length 0
IP 10.0.0.1.56524 > 10.0.0.2.5001: Flags [.], seq 598049:600945, ack 1, win 58, options [nop,nop,TS val 10942400 ecr 10942399], length 2896
IP 10.0.0.2.5001 > 10.0.0.1.56524: Flags [.], ack 596601, win 838, options [nop,nop,TS val 10942412 ecr 10942388,nop,nop,sack 1
{598049:600945}], length 0
IP 10.0.0.1.56524 > 10.0.0.2.5001: Flags [.], seq 600945:603841, ack 1, win 58, options [nop,nop,TS val 10942403 ecr 10942402], length 2896
IP 10.0.0.2.5001 > 10.0.0.1.56524: Flags [.], ack 596601, win 838, options [nop,nop,TS val 10942418 ecr 10942388,nop,nop,sack 1
{598049:603841}], length 0
IP 10.0.0.1.56524 > 10.0.0.2.5001: Flags [.], seq 603841:606737, ack 1, win 58, options [nop,nop,TS val 10942412 ecr 10942412], length 2896
IP 10.0.0.2.5001 > 10.0.0.1.56524: Flags [.], ack 596601, win 838, options [nop,nop,TS val 10942424 ecr 10942388,nop,nop,sack 1
{598049:606737}], length 0
IP 10.0.0.1.56524 > 10.0.0.2.5001: Flags [.], seq 606737:608185, ack 1, win 58, options [nop,nop,TS val 10942412 ecr 10942412], length 1448
IP 10.0.0.2.5001 > 10.0.0.1.56524: Flags [.], ack 596601, win 838, options [nop,nop,TS val 10942430 ecr 10942388,nop,nop,sack 1
```

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

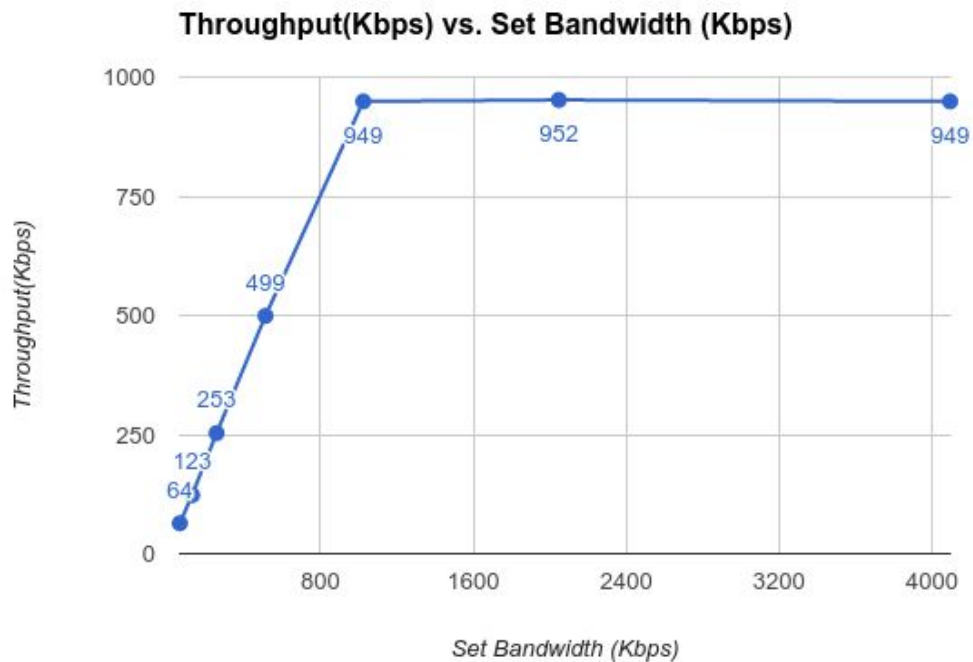
So here too we can see there are ACK, PUSH, FIN, SYN packets primarily and some packets which are combination of some of the above four flags like FIN-PUSH-ACK.

2. A)

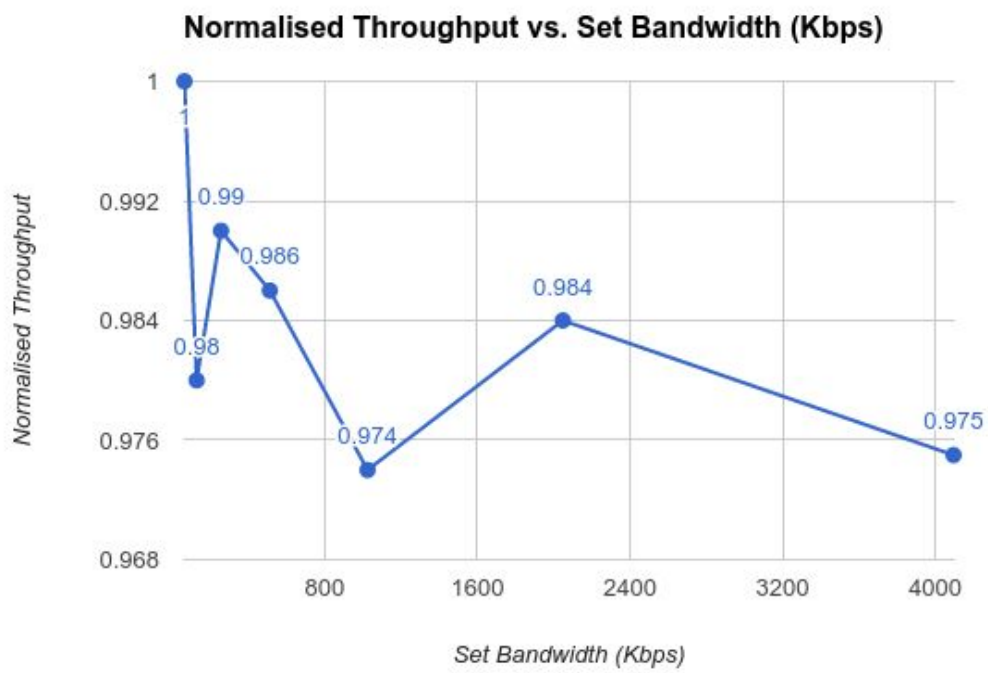
The normalised TCP goodput is as follows :

$$\begin{aligned}\text{Normalized TCP Goodput} &= \text{total amount of data received} / \text{total amount of data sent} \\ &= 1571105/1571105 \\ &= 1\end{aligned}$$

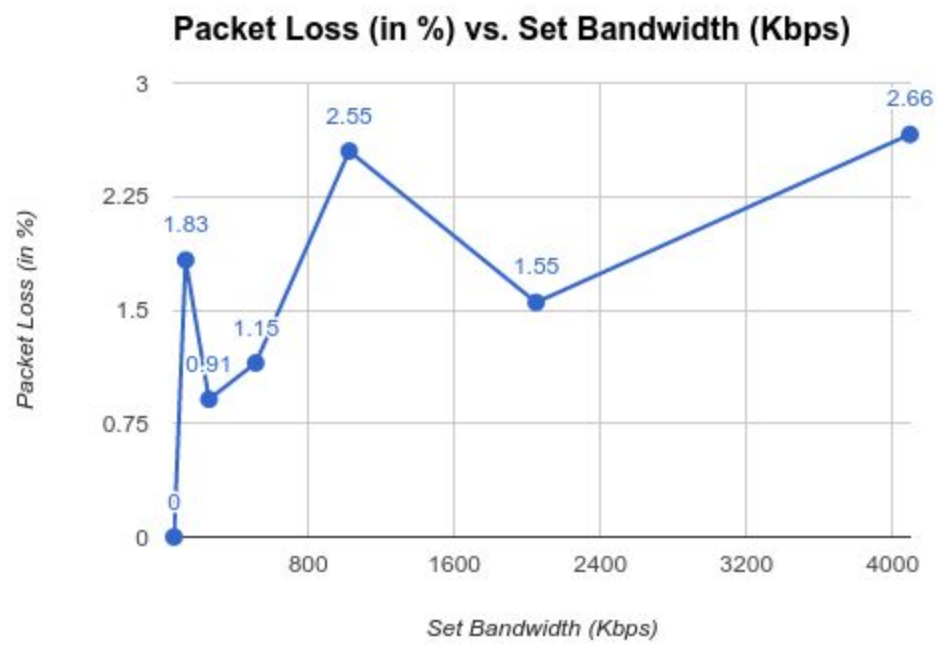
B)



C)



D)



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

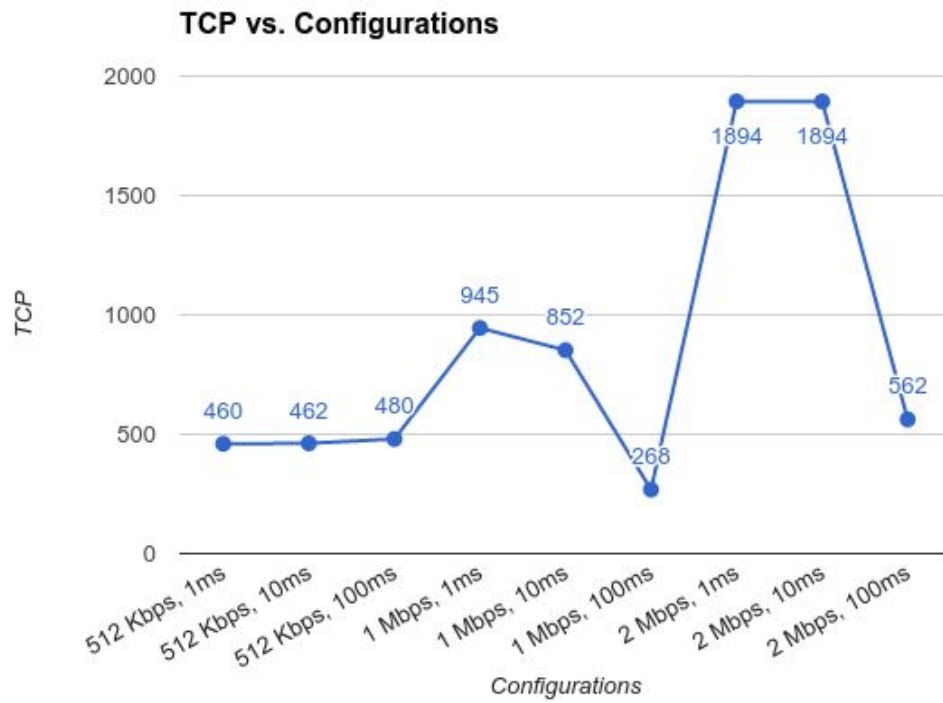
*****For normalised throughput calculation we find the total sent packets by sender and total received packets by the receiver. So, normalised throughput = received packets/ sent packets.

*****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 :

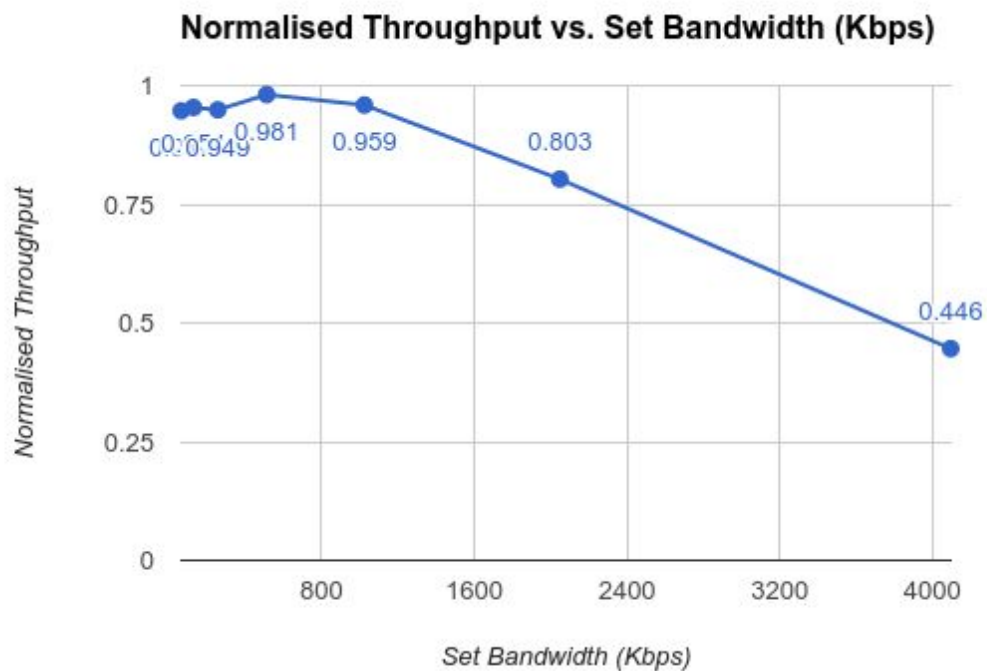
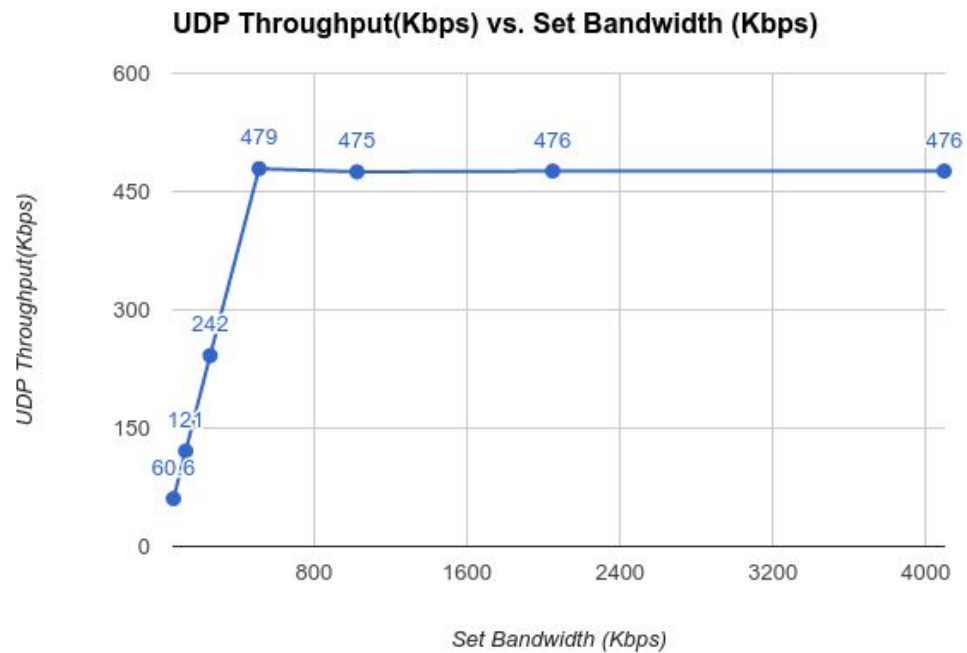
The TCP bandwidth observed for every of the nine configurations of the middle (between two switches) link :

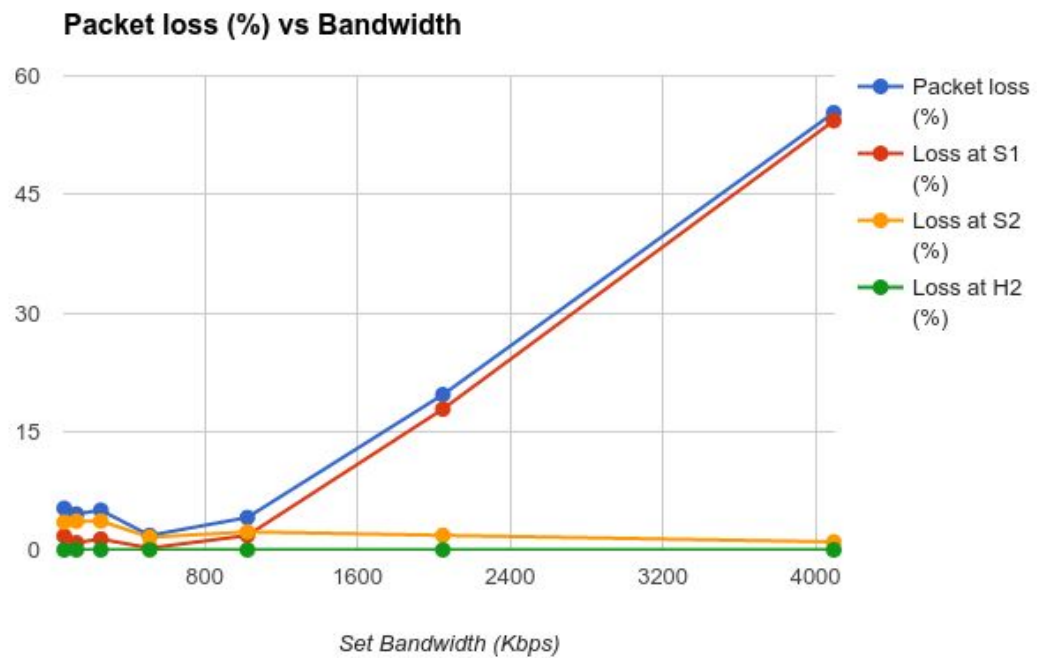


1. Bandwidth = 512 Kbps, Delay = 1 ms

The normalised TCP goodput is as follows :

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

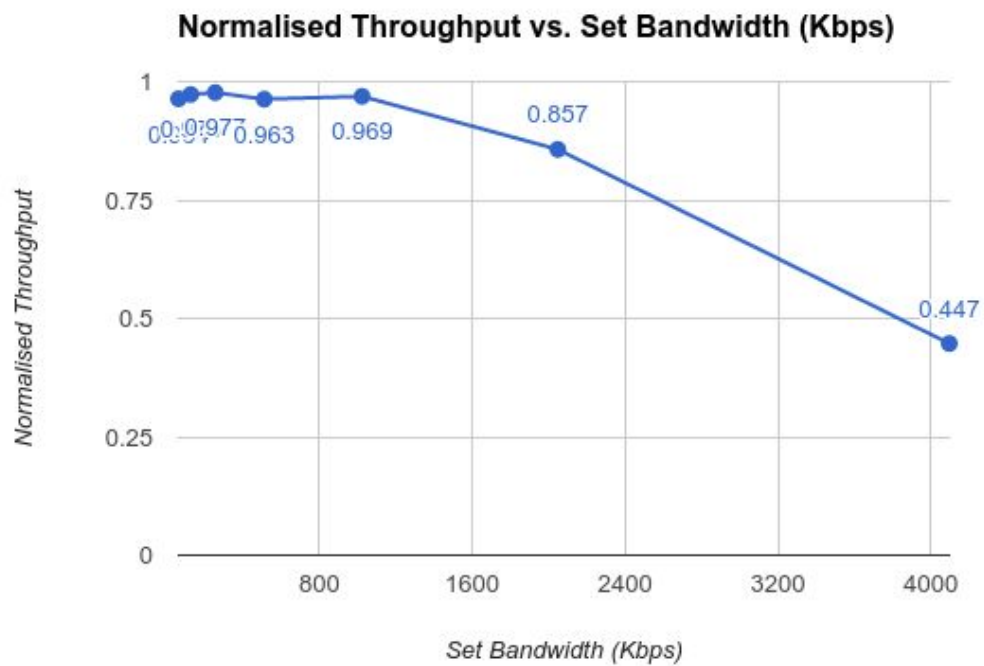
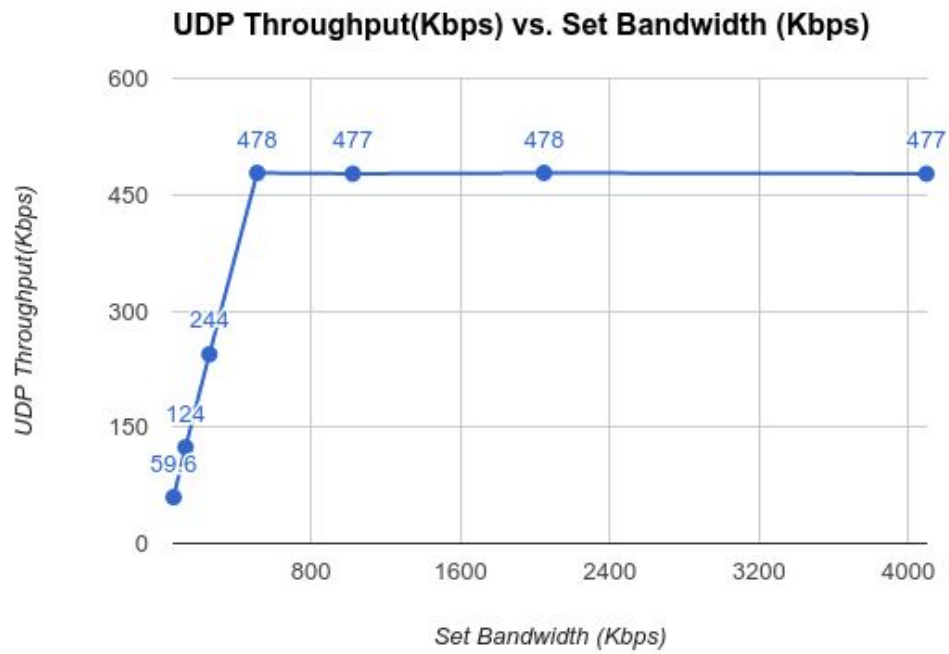


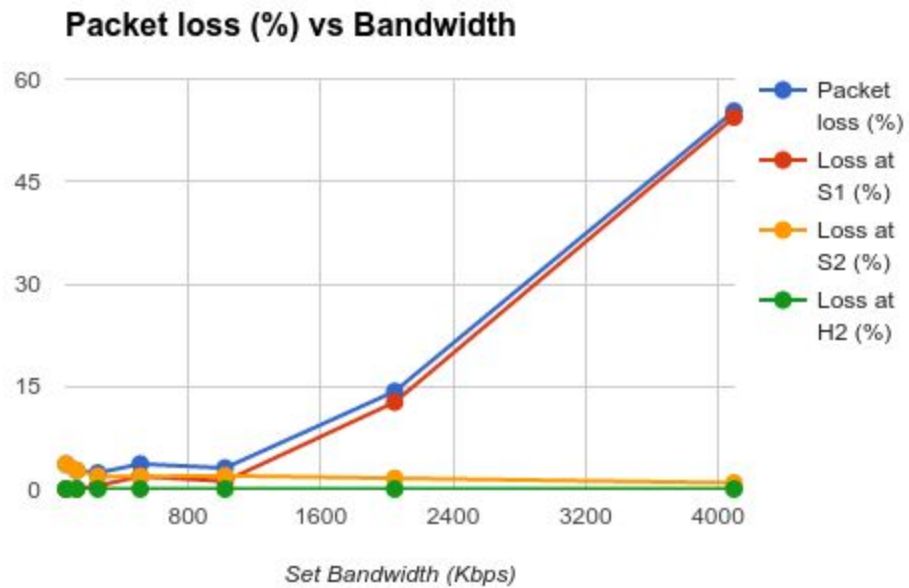


2.Bandwidth = 512 Kbps, Delay = 10 ms

The normalised TCP goodput is as follows :

$$\begin{aligned}
 \text{Normalized TCP Goodput} &= \text{total amount of data received} / \text{total amount of data sent} \\
 &= 786457/786457 \\
 &= 1
 \end{aligned}$$

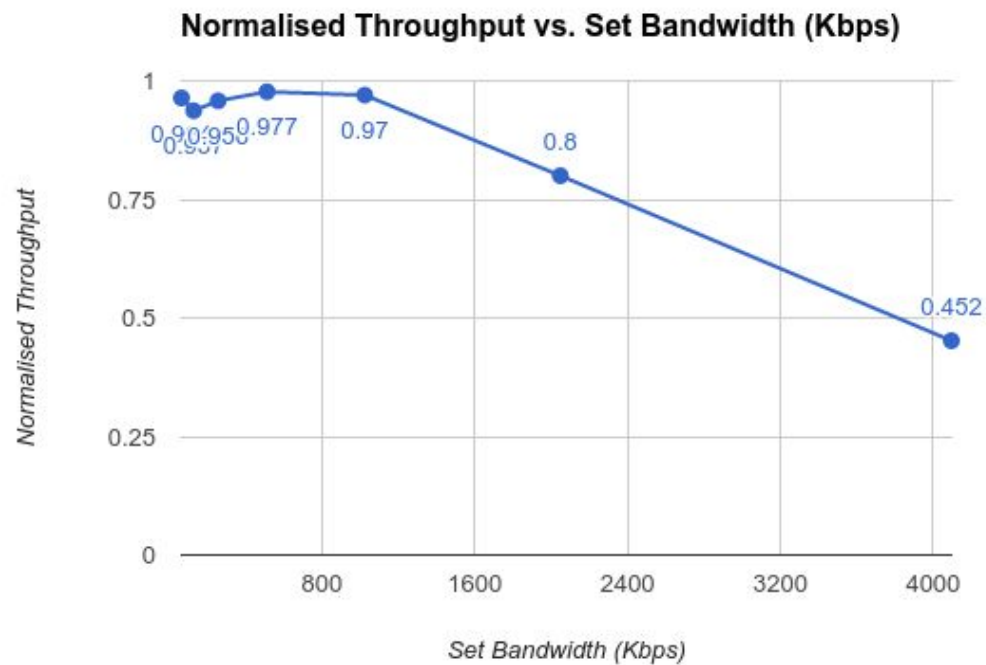
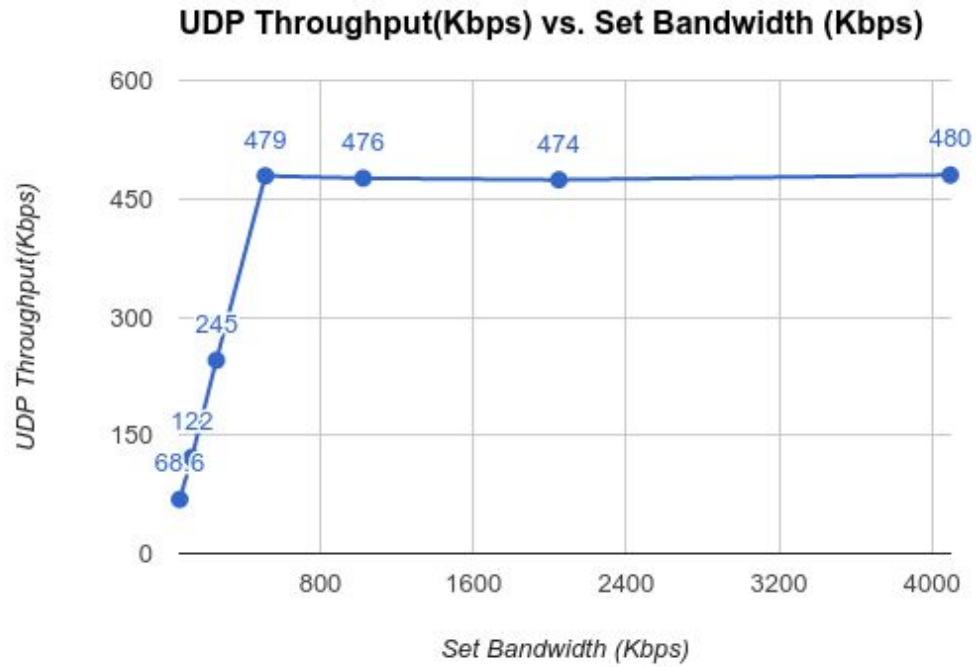


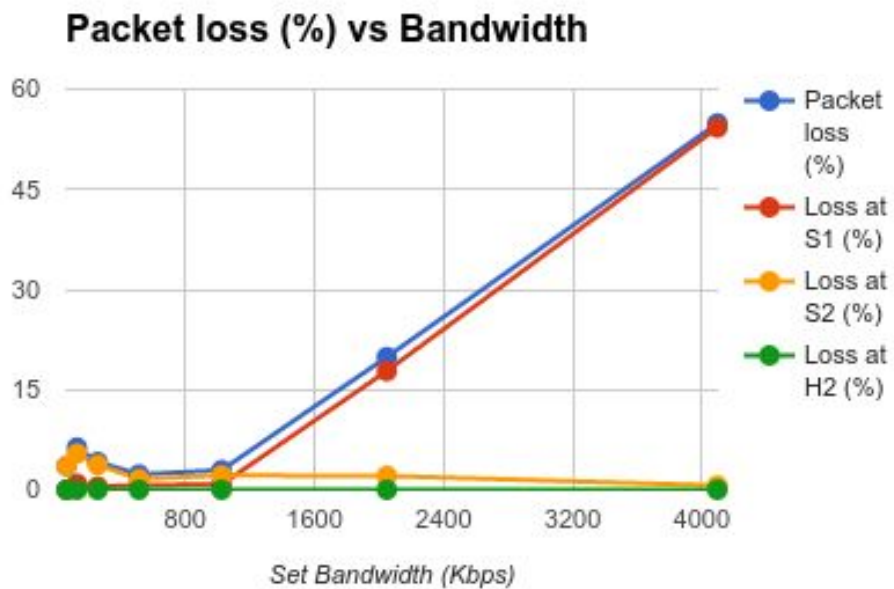


3. Bandwidth = 512 Kbps, Delay = 100 ms

The normalised TCP goodput is as follows :

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

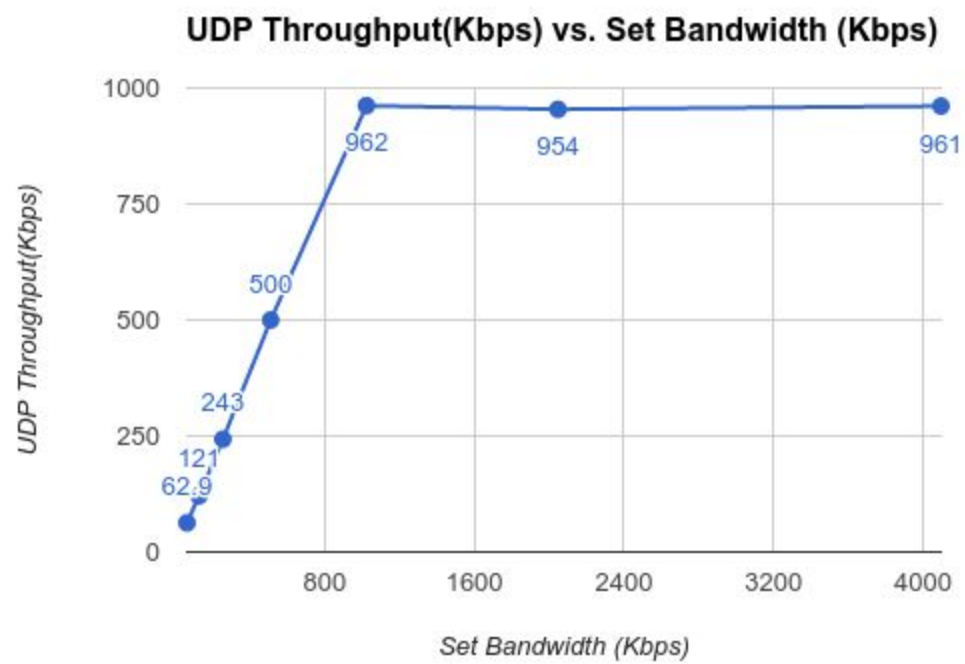


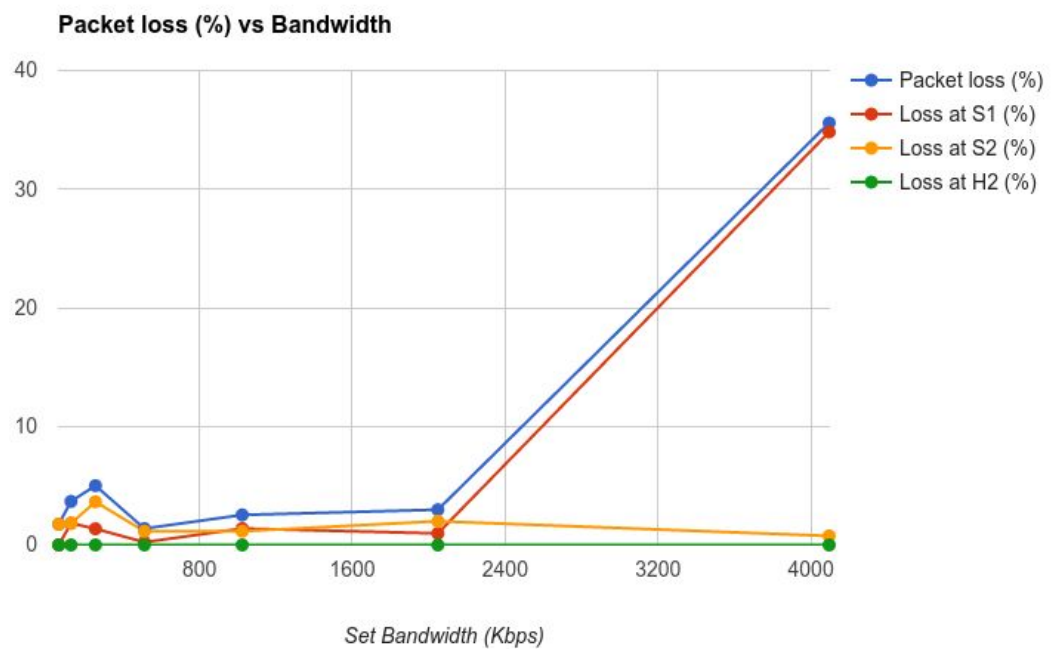
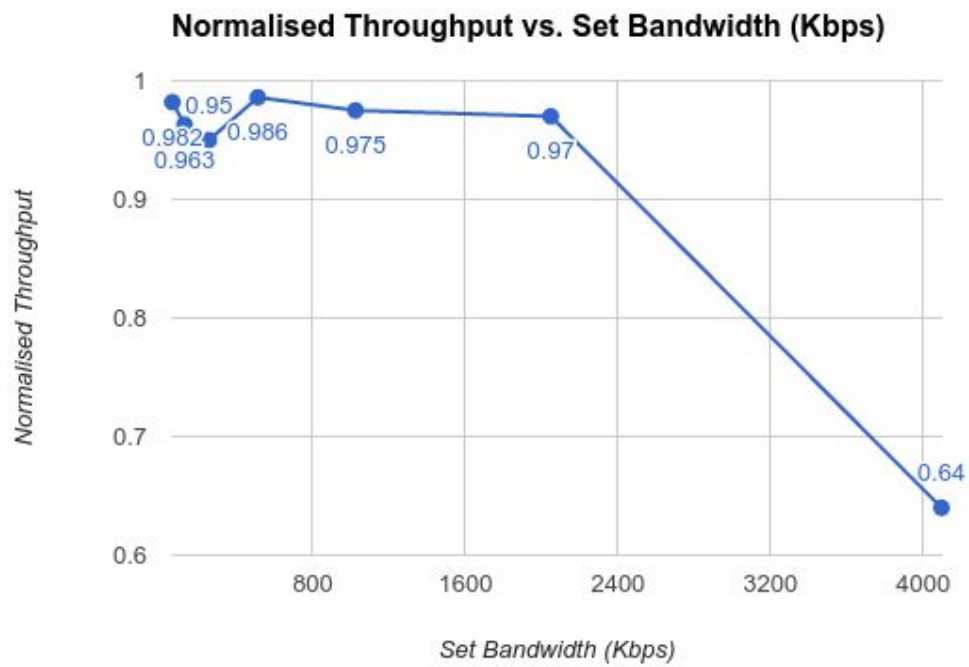


4. Bandwidth = 512 Kbps, Delay = 1 ms

The normalised TCP goodput is as follows :

$$\begin{aligned}
 \text{Normalized TCP Goodput} &= \text{total amount of data received} / \text{total amount of data sent} \\
 &= 1441817 / 1441817 \\
 &= 1
 \end{aligned}$$

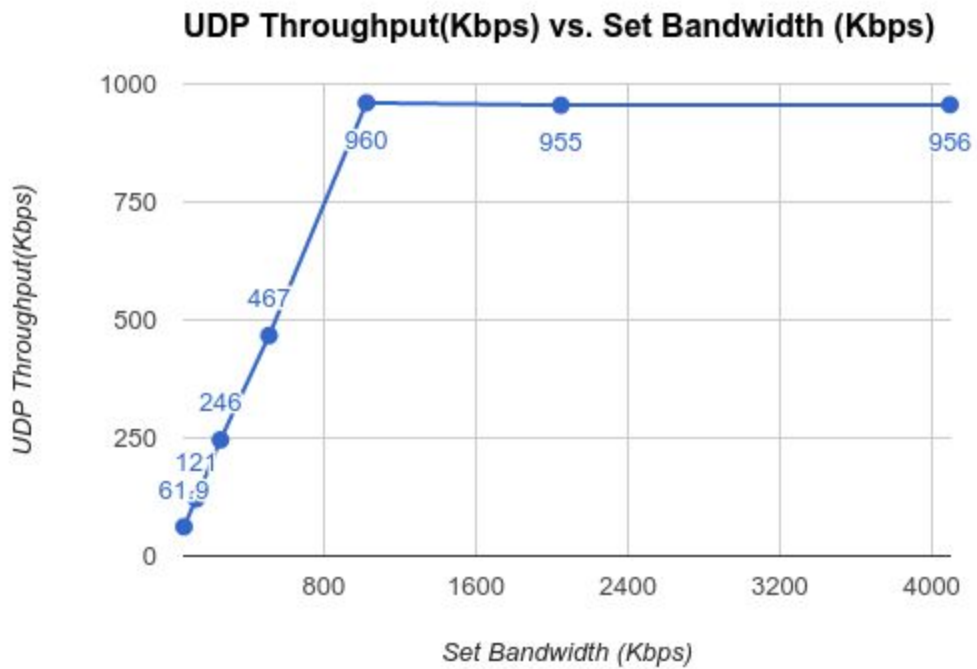


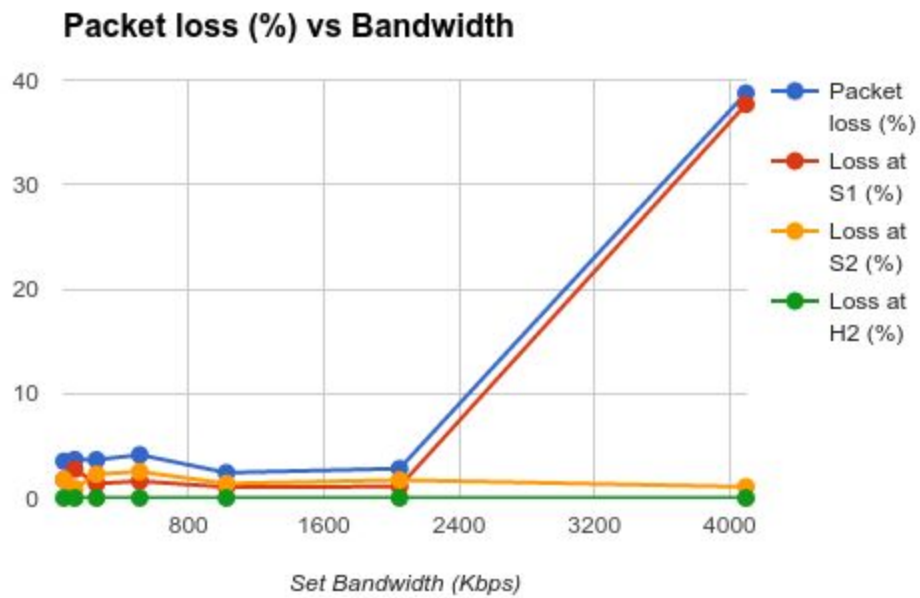
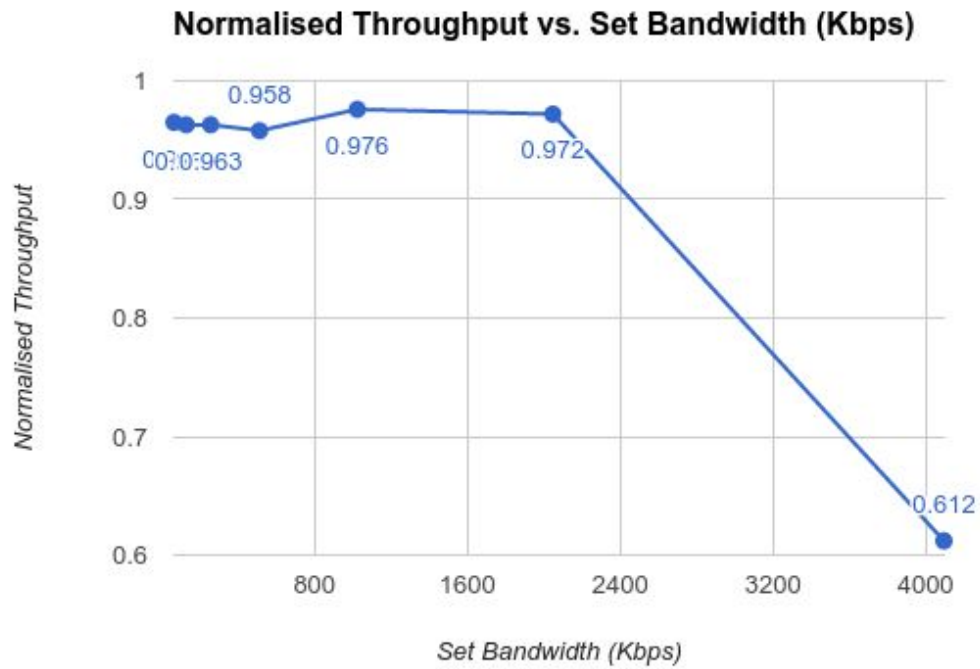


5. Bandwidth = 1 Mbps, Delay = 10 ms

The normalised TCP goodput is as follows :

$$\begin{aligned}\text{Normalized TCP Goodput} &= \text{total amount of data received} / \text{total amount of data sent} \\ &= 1310745 / 1310745 \\ &= 1\end{aligned}$$

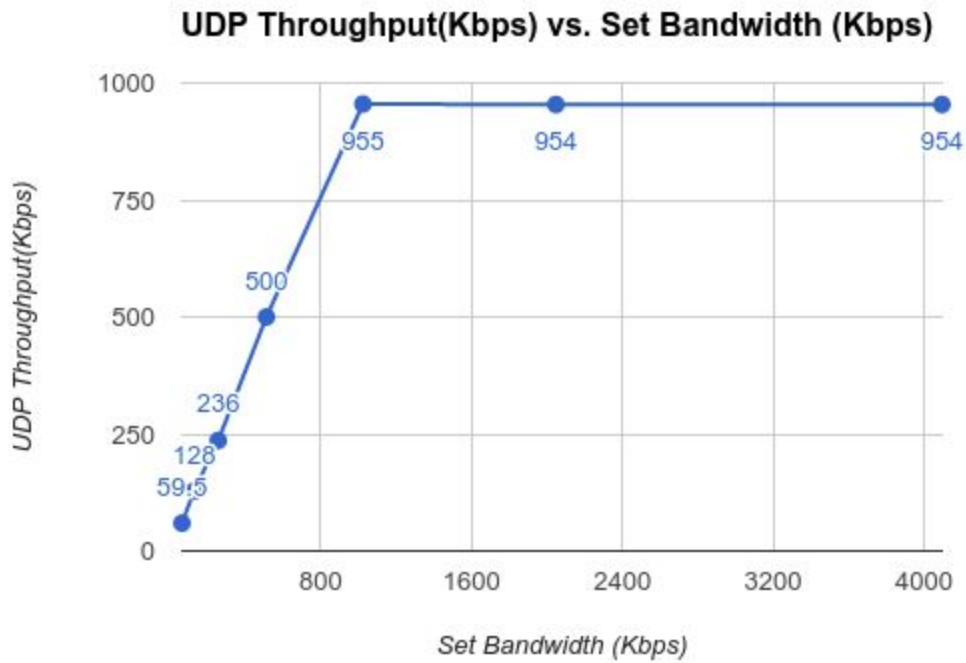


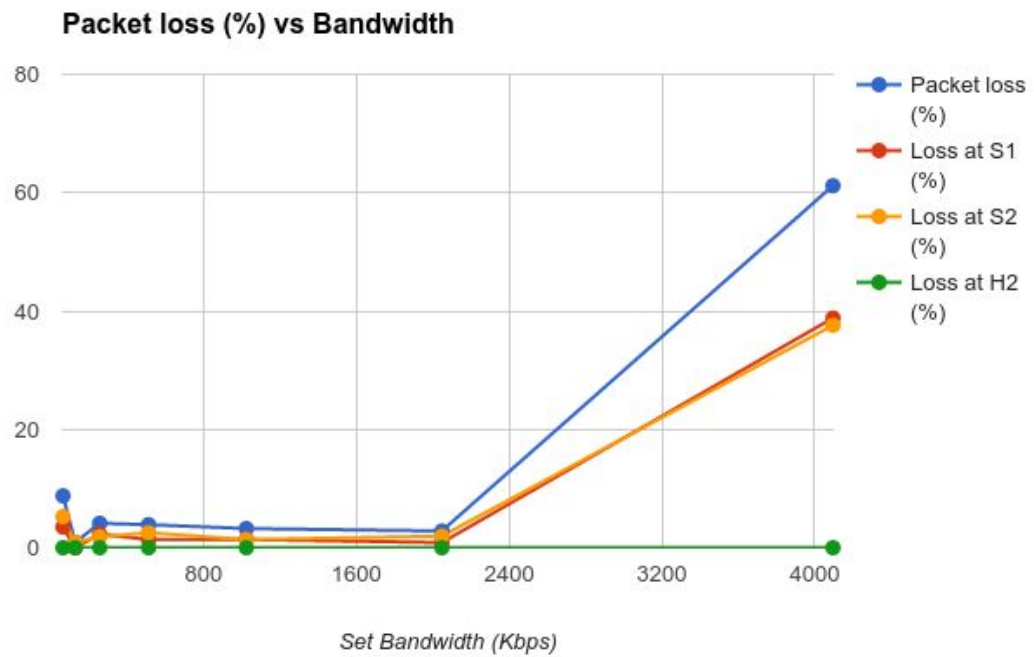
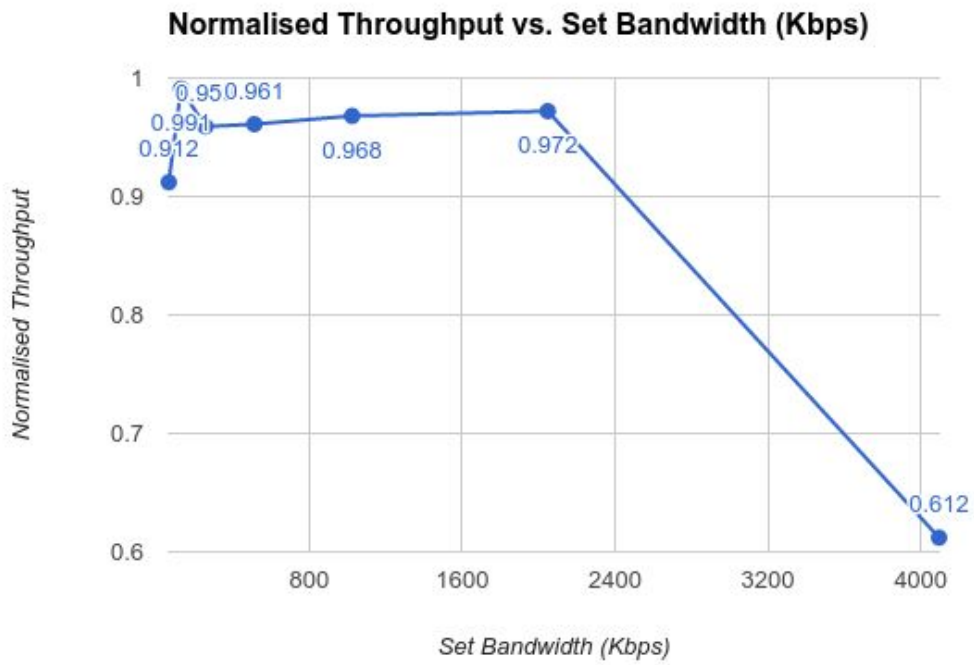


6. Bandwidth = 1 Mbps, Delay = 100 ms

The normalised TCP goodput is as follows :

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

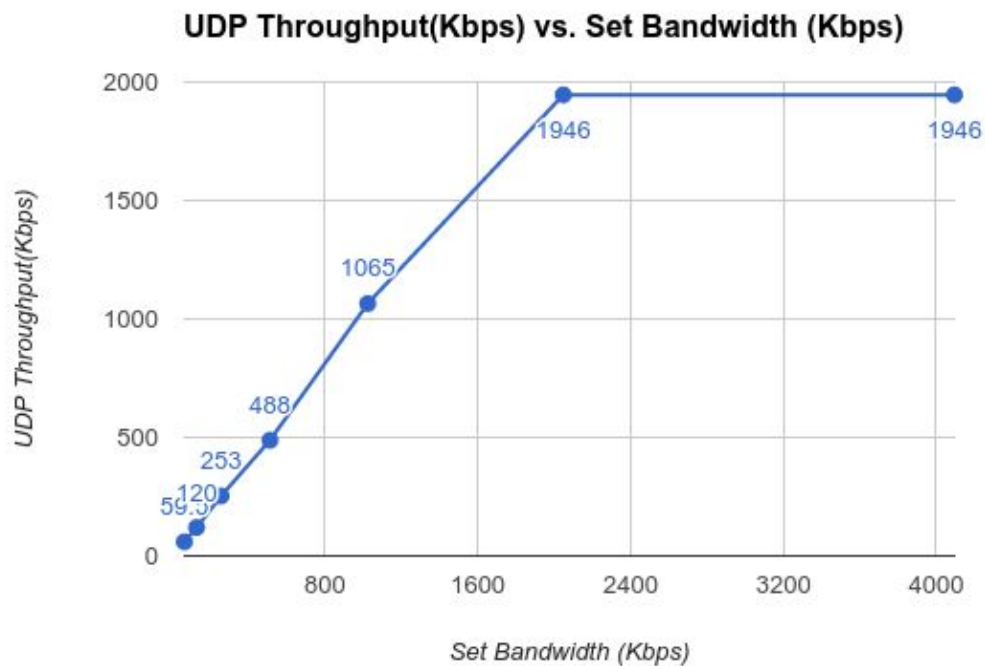


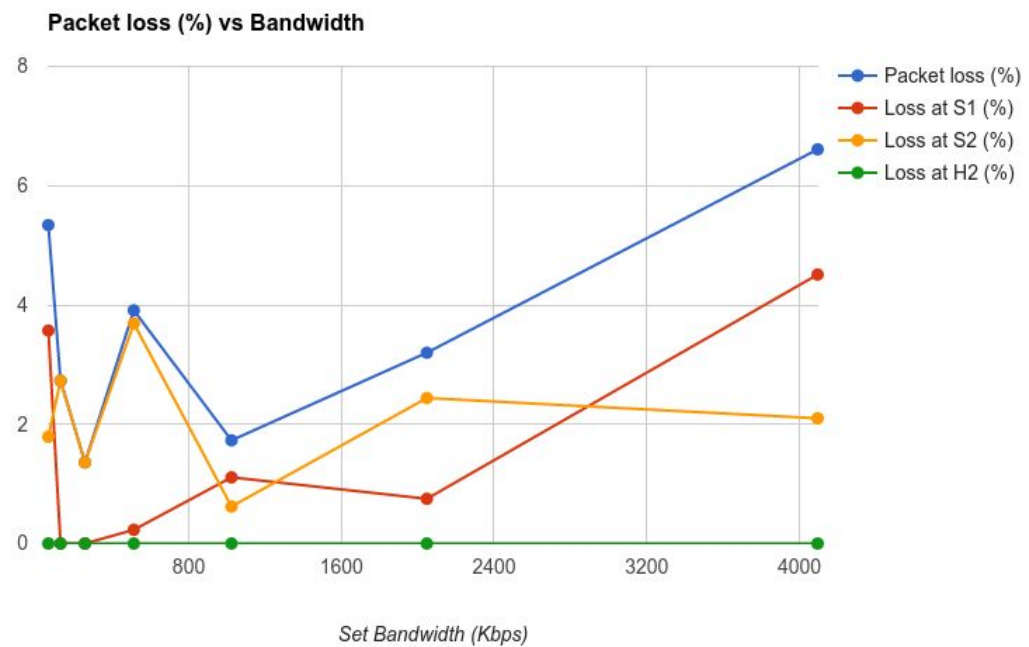
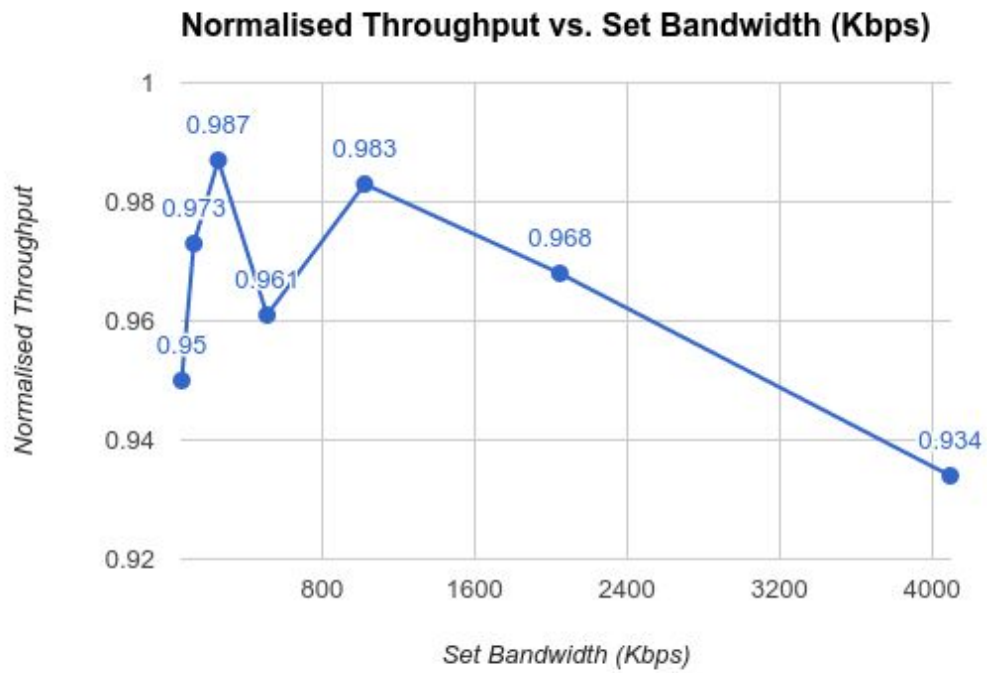


7. Bandwidth = 2 Mbps, Delay = 1 ms

The normalised TCP goodput is as follows :

$$\begin{aligned}\text{Normalized TCP Goodput} &= \text{total amount of data received} / \text{total amount of data sent} \\ &= 2621465 / 2621465 \\ &= 1\end{aligned}$$

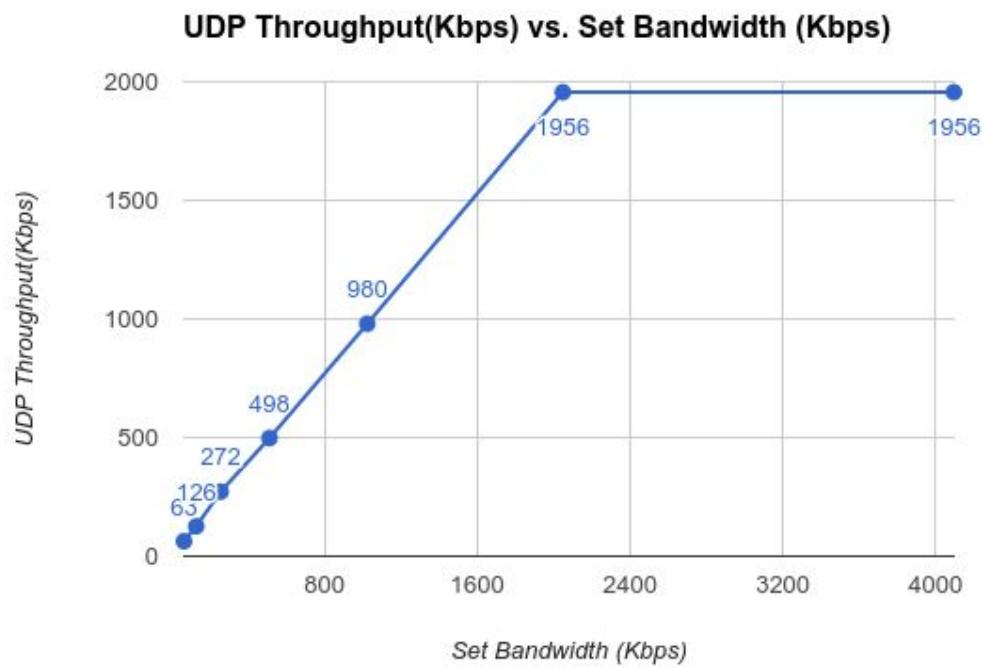


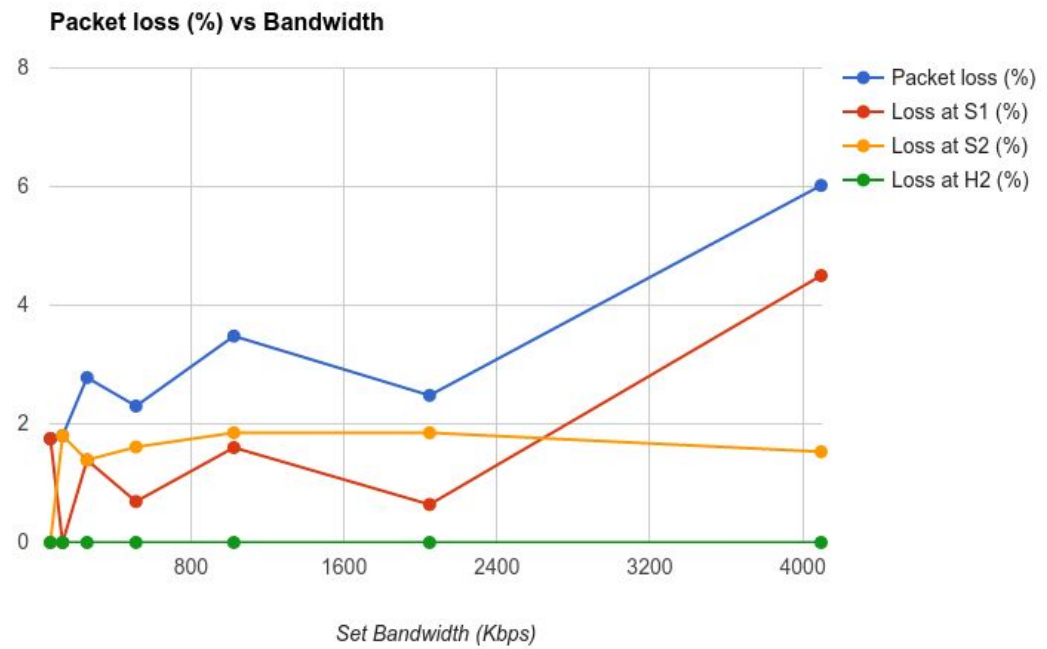
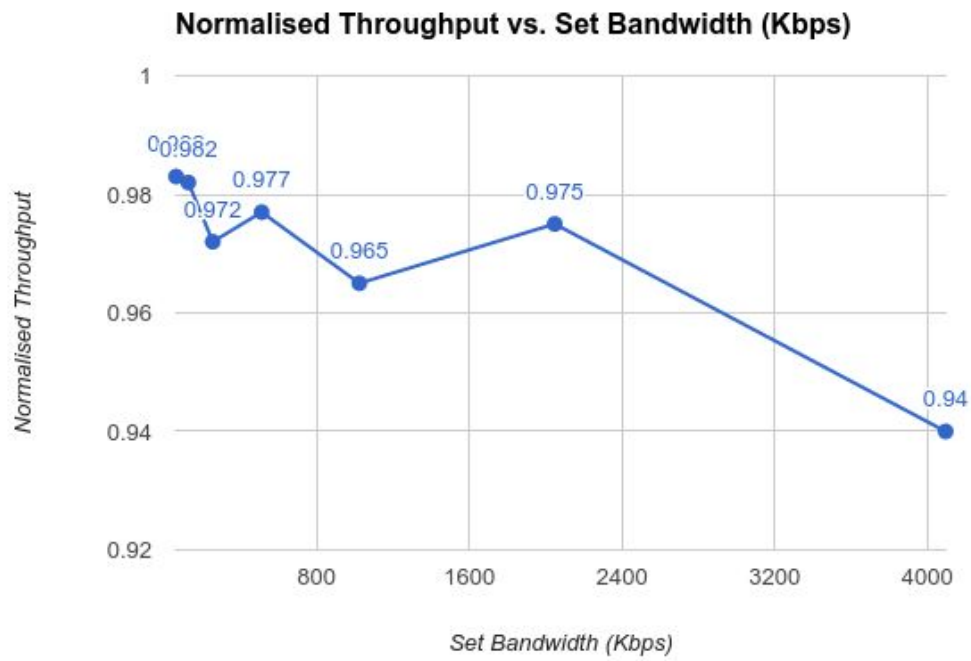


8. Bandwidth = 2 Mbps, Delay = 10 ms

The normalised TCP goodput is as follows :

$$\begin{aligned}\text{Normalized TCP Goodput} &= \text{total amount of data received} / \text{total amount of data sent} \\ &= 2751225/2751225 \\ &= 1\end{aligned}$$

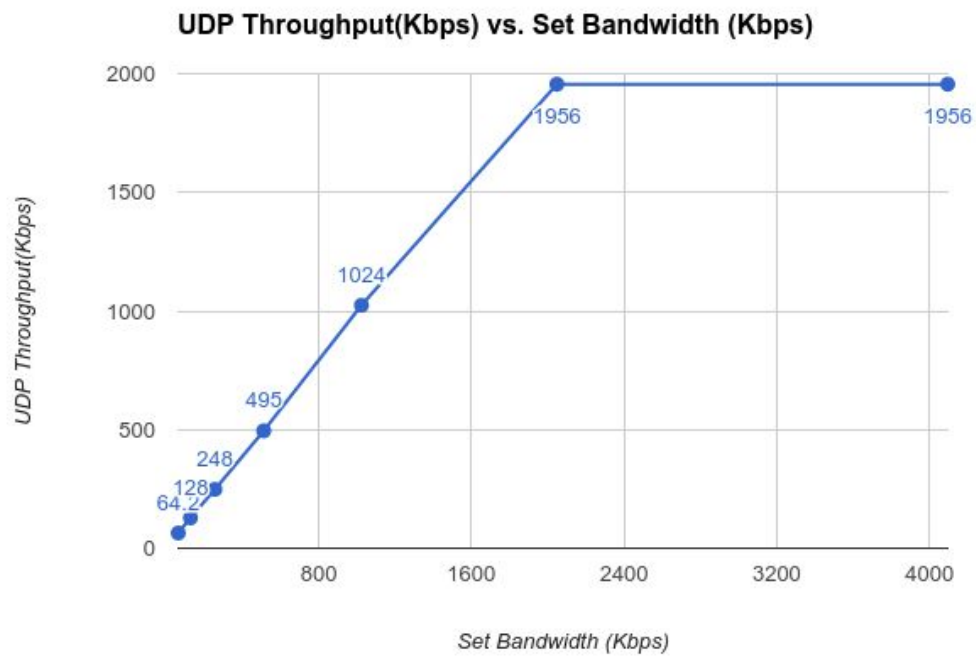


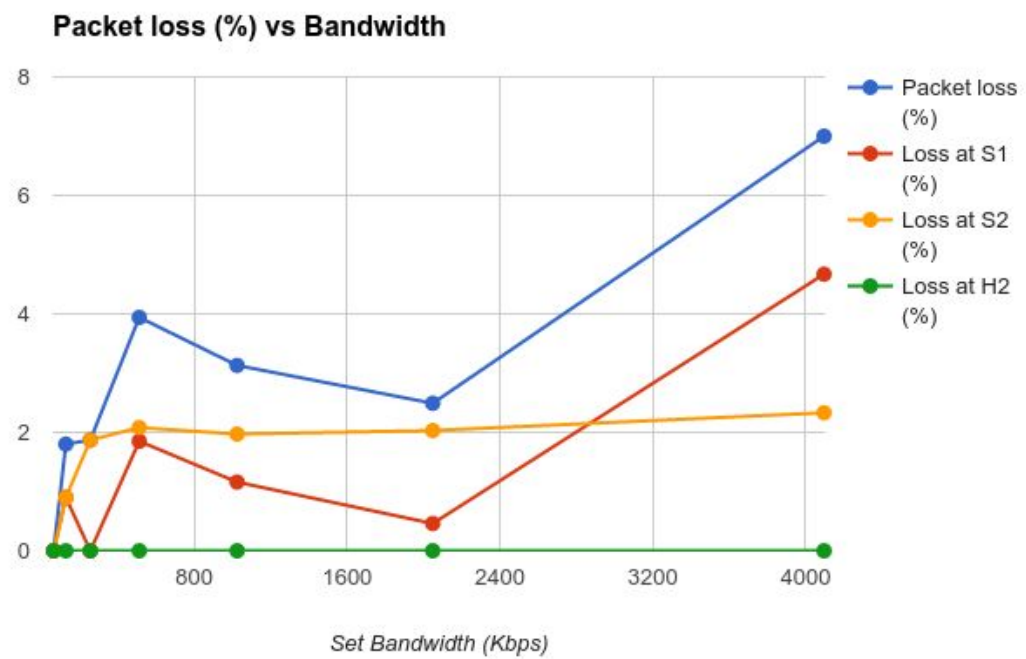
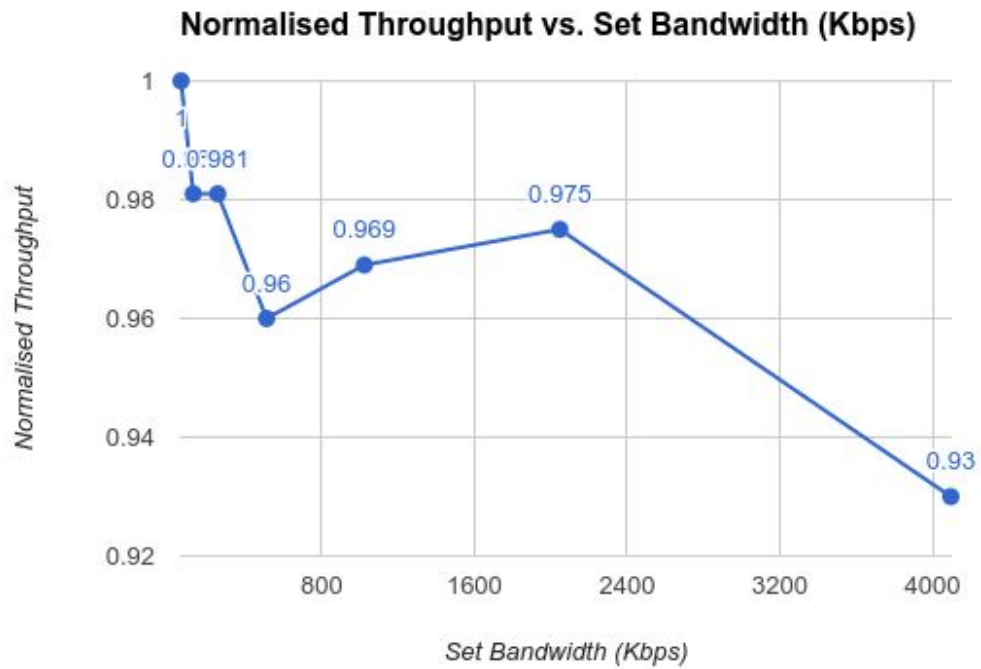


9. Bandwidth = 2 Mbps, Delay = 100 ms

The normalised TCP goodput is as follows :

$$\begin{aligned}\text{Normalized TCP Goodput} &= \text{total amount of data received} / \text{total amount of data sent} \\ &= 1048377 / 1048377 \\ &= 1\end{aligned}$$





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.

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