Name: Darshi Gaurang Doshi (21110050)
Name: Disha Chopra (21110057)

CS433 Assignment-2

Github Portal Link - https://github.com/DarshiDoshi/CN_Assignment-2

Question 1

(a)

The above commands were used to add the routing paths connecting the routers.

    info(net['r1'].cmd("ip route add 10.1.0.0/24 via 10.100.0.2 dev r12"))#12
    info(net['r2'].cmd("ip route add 10.0.0.0/24 via 10.100.0.1 dev r21"))#21
    info(net['r3'].cmd("ip route add 10.0.0.0/24 via 10.101.0.3 dev r31"))#31
    info(net['r1'].cmd("ip route add 10.2.0.0/24 via 10.101.0.4 dev r13"))#13
    info(net['r2'].cmd("ip route add 10.2.0.0/24 via 10.102.0.6 dev r23"))#23
    info(net['r3'].cmd("ip route add 10.1.0.0/24 via 10.102.0.5 dev r32"))#32

This part of the code uses the cmd method to execute a command on a router.
ip route add 10.2.0.0/24 via 10.101.0.4 dev r13: adds a route to the routing table. It specifies that traffic destined for the IP subnet 10.2.0.0/24 should be sent via the next hop address 10.101.0.4 and the outgoing network interface r13.

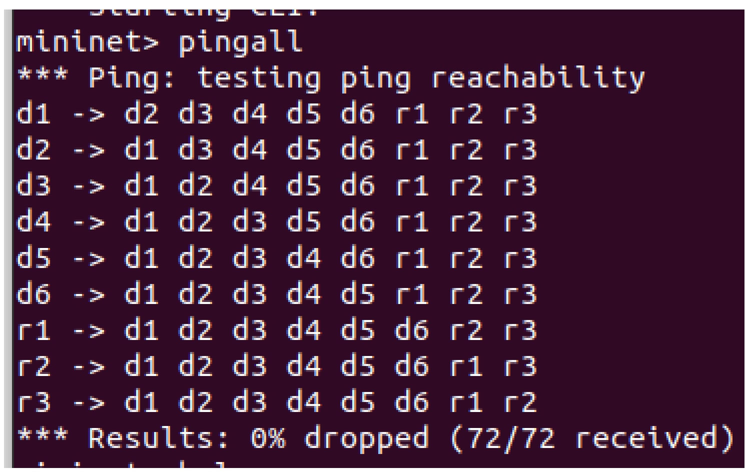
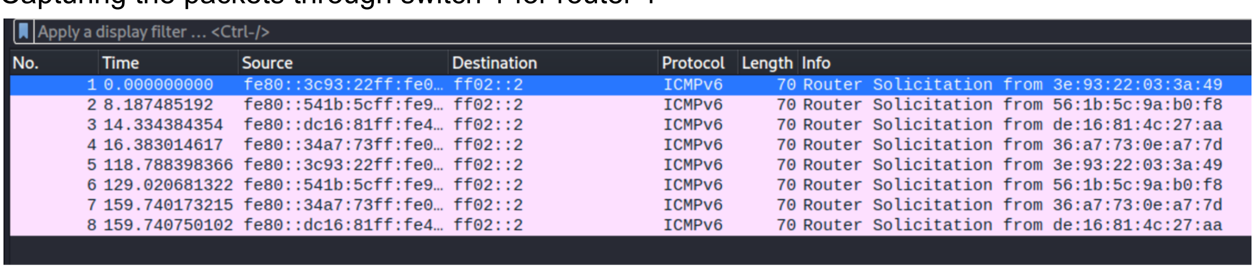


Fig 1.1

(b)

Capturing the packets through switch 1 for router 1



Apply a display filter … <Ctrl-/>

No.	Time	Source	Destination	Protocol	Length	Info
1	0.000000000	fe80::3c93:22ff:fe0…	ff02::2	ICMPv6	70	Router Solicitation from 3e:93:22:03:3a:49
2	8.187485192	fe80::541b:5cff:fe9…	ff02::2	ICMPv6	70	Router Solicitation from 56:1b:5c:9a:b0:f8
3	14.334384354	fe80::dc16:81ff:fe4…	ff02::2	ICMPv6	70	Router Solicitation from de:16:81:4c:27:aa
4	16.383014617	fe80::34a7:73ff:fe0…	ff02::2	ICMPv6	70	Router Solicitation from 36:a7:73:0e:a7:7d
5	118.788398366	fe80::3c93:22ff:fe0…	ff02::2	ICMPv6	70	Router Solicitation from 3e:93:22:03:3a:49
6	129.020681322	fe80::541b:5cff:fe9…	ff02::2	ICMPv6	70	Router Solicitation from 56:1b:5c:9a:b0:f8
7	159.740173215	fe80::34a7:73ff:fe0…	ff02::2	ICMPv6	70	Router Solicitation from 36:a7:73:0e:a7:7d
8	159.740750102	fe80::dc16:81ff:fe4…	ff02::2	ICMPv6	70	Router Solicitation from de:16:81:4c:27:aa

(c)

```
mininet> d1 traceroute d6
traceroute to 10.2.0.252 (10.2.0.252), 30 hops max, 60 byte packets
 1  10.0.0.1 (10.0.0.1)  4.320 ms  3.862 ms  3.834 ms
 2  10.101.0.4 (10.101.0.4)  3.823 ms  3.810 ms  3.802 ms
 3  10.2.0.252 (10.2.0.252)  6.632 ms  6.611 ms  6.561 ms
```

Fig 1.3 Default Route (d1 → r1 → r3 → d6)

```
s1 s2 s3 ...
*** Starting CLI:
mininet> d1 traceroute d6
traceroute to 10.2.0.252 (10.2.0.252), 30 hops max, 60 byte packets
 1  10.0.0.1 (10.0.0.1)  5.098 ms  5.075 ms  5.090 ms
 2  10.100.0.2 (10.100.0.2)  5.098 ms  5.102 ms  5.109 ms
 3  10.101.0.4 (10.101.0.4)  5.115 ms  5.120 ms  5.138 ms
 4  10.2.0.252 (10.2.0.252)  12.973 ms  12.980 ms  12.986 ms
```

Fig 1.4 Changed Route (d1 → r1 → r2 → r3 → d6)

→ **iperf**

Before changing the default route -

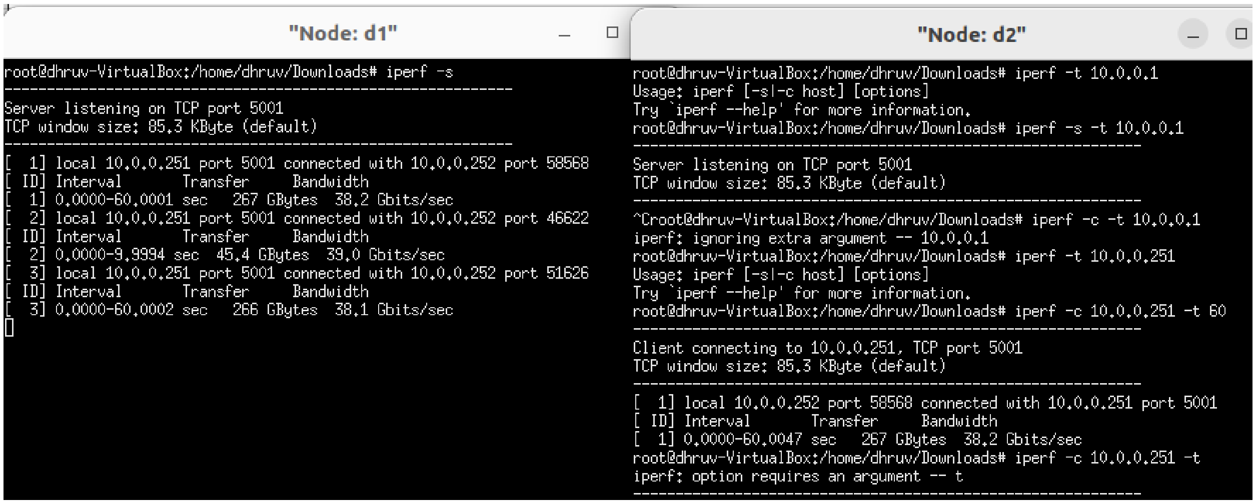
"Node: d1"

```
root@dhruv-VirtualBox:/home/dhruv/Downloads# iperf -s
------------------------------------------------------------
Server listening on TCP port 5001
TCP window size: 85.3 KByte (default)
------------------------------------------------------------
[  1] local 10.0.0.251 port 5001 connected with 10.0.0.252 port 58568
[ ID] Interval       Transfer     Bandwidth
[  1] 0.0000-60.0001 sec   267 GBytes  38.2 Gbits/sec
[  2] local 10.0.0.251 port 5001 connected with 10.0.0.252 port 46622
[ ID] Interval       Transfer     Bandwidth
[  2] 0.0000-9.9994 sec  45.4 GBytes  39.0 Gbits/sec
[  3] local 10.0.0.251 port 5001 connected with 10.0.0.252 port 51626
[ ID] Interval       Transfer     Bandwidth
[  3] 0.0000-60.0002 sec   266 GBytes  38.1 Gbits/sec
```

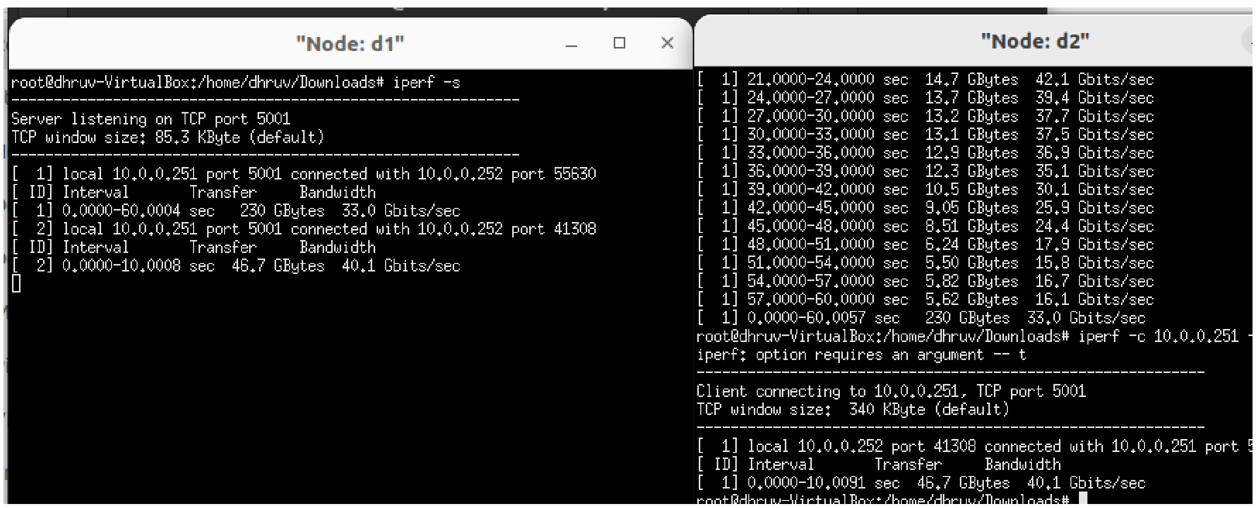
"Node: d2"

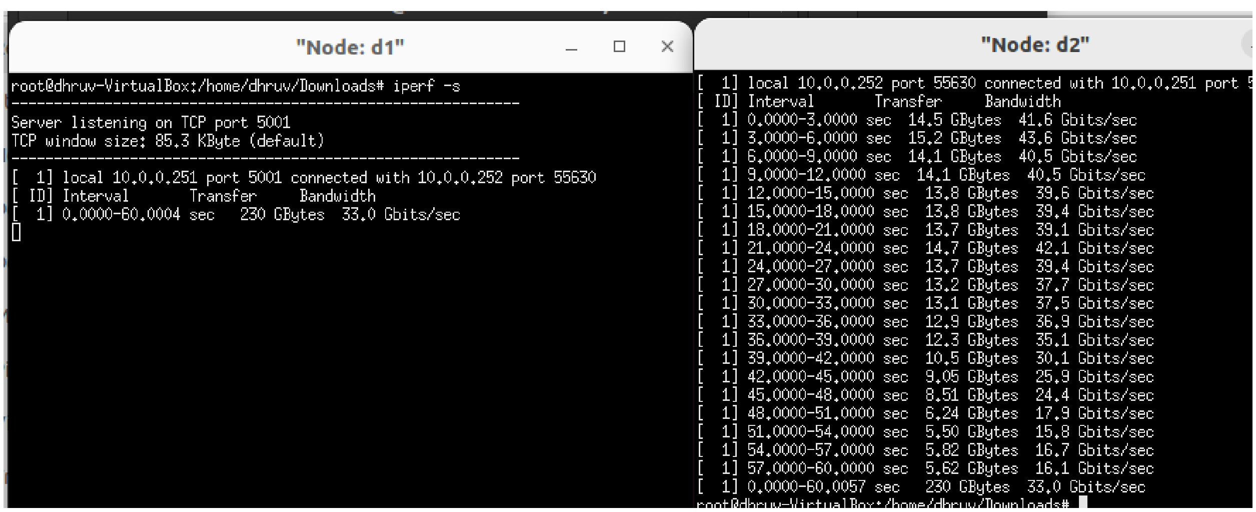
```
[  1] local 10.0.0.252 port 51626 connected with 10.0.0.251 port 5001
[ ID] Interval       Transfer     Bandwidth
[  1] 0.0000-3.0000 sec  15.5 GBytes  44.5 Gbits/sec
[  1] 3.0000-6.0000 sec  16.0 GBytes  45.7 Gbits/sec
[  1] 6.0000-9.0000 sec  13.9 GBytes  39.9 Gbits/sec
[  1] 9.0000-12.0000 sec  13.2 GBytes  37.8 Gbits/sec
[  1] 12.0000-15.0000 sec  13.6 GBytes  38.9 Gbits/sec
[  1] 15.0000-18.0000 sec  13.5 GBytes  38.6 Gbits/sec
[  1] 18.0000-21.0000 sec  13.1 GBytes  37.6 Gbits/sec
[  1] 21.0000-24.0000 sec  14.6 GBytes  41.9 Gbits/sec
[  1] 24.0000-27.0000 sec  16.0 GBytes  45.7 Gbits/sec
[  1] 27.0000-30.0000 sec  9.49 GBytes  27.2 Gbits/sec
[  1] 30.0000-33.0000 sec  11.8 GBytes  33.7 Gbits/sec
[  1] 33.0000-36.0000 sec  13.1 GBytes  37.5 Gbits/sec
[  1] 36.0000-39.0000 sec  11.5 GBytes  32.8 Gbits/sec
[  1] 39.0000-42.0000 sec  11.8 GBytes  33.8 Gbits/sec
[  1] 42.0000-45.0000 sec  13.2 GBytes  37.8 Gbits/sec
[  1] 45.0000-48.0000 sec  13.4 GBytes  38.4 Gbits/sec
[  1] 48.0000-51.0000 sec  13.7 GBytes  39.1 Gbits/sec
[  1] 51.0000-54.0000 sec  13.6 GBytes  38.8 Gbits/sec
[  1] 54.0000-57.0000 sec  12.1 GBytes  34.6 Gbits/sec
[  1] 57.0000-60.0000 sec  13.5 GBytes  38.8 Gbits/sec
[  1] 0.0000-60.0081 sec   266 GBytes  38.1 Gbits/sec
root@dhruv-VirtualBox:/home/dhruv/Downloads#
```

Fig 1.5



After changing the default route -





By doing iperf, it can be seen that the transfer and bandwidth have decreased as we changed the default path of the routers.

→ **ping**

Before changing the default route -

```
mininet> d1 ping d6
PING 10.2.0.252 (10.2.0.252) 56(84) bytes of data.
64 bytes from 10.2.0.252: icmp_seq=1 ttl=62 time=4.40 ms
64 bytes from 10.2.0.252: icmp_seq=2 ttl=62 time=0.350 ms
64 bytes from 10.2.0.252: icmp_seq=3 ttl=62 time=0.106 ms
64 bytes from 10.2.0.252: icmp_seq=4 ttl=62 time=0.097 ms
64 bytes from 10.2.0.252: icmp_seq=5 ttl=62 time=0.091 ms
64 bytes from 10.2.0.252: icmp_seq=6 ttl=62 time=0.078 ms
64 bytes from 10.2.0.252: icmp_seq=7 ttl=62 time=0.096 ms
64 bytes from 10.2.0.252: icmp_seq=8 ttl=62 time=0.077 ms
64 bytes from 10.2.0.252: icmp_seq=9 ttl=62 time=0.077 ms
64 bytes from 10.2.0.252: icmp_seq=10 ttl=62 time=0.089 ms
64 bytes from 10.2.0.252: icmp_seq=11 ttl=62 time=0.107 ms
64 bytes from 10.2.0.252: icmp_seq=12 ttl=62 time=0.104 ms
64 bytes from 10.2.0.252: icmp_seq=13 ttl=62 time=0.084 ms
64 bytes from 10.2.0.252: icmp_seq=14 ttl=62 time=0.145 ms
64 bytes from 10.2.0.252: icmp_seq=15 ttl=62 time=0.076 ms
64 bytes from 10.2.0.252: icmp_seq=16 ttl=62 time=0.109 ms
64 bytes from 10.2.0.252: icmp_seq=17 ttl=62 time=0.085 ms
64 bytes from 10.2.0.252: icmp_seq=18 ttl=62 time=0.096 ms
64 bytes from 10.2.0.252: icmp_seq=19 ttl=62 time=0.084 ms
64 bytes from 10.2.0.252: icmp_seq=20 ttl=62 time=0.103 ms
64 bytes from 10.2.0.252: icmp_seq=21 ttl=62 time=0.082 ms
64 bytes from 10.2.0.252: icmp_seq=22 ttl=62 time=0.079 ms
64 bytes from 10.2.0.252: icmp_seq=23 ttl=62 time=0.105 ms
^C
--- 10.2.0.252 ping statistics ---
23 packets transmitted, 23 received, 0% packet loss, time 22499ms
rtt min/avg/max/mdev = 0.076/0.292/4.404/0.878 ms
```

After changing the default route -

```
rtt min/avg/max/mdev = 0.069/0.158/2.371/0.336 ms
mininet> d1 ping d6
PING 10.2.0.252 (10.2.0.252) 56(84) bytes of data.
64 bytes from 10.2.0.252: icmp_seq=1 ttl=62 time=2.51 ms
64 bytes from 10.2.0.252: icmp_seq=2 ttl=62 time=2.75 ms
64 bytes from 10.2.0.252: icmp_seq=3 ttl=62 time=0.492 ms
64 bytes from 10.2.0.252: icmp_seq=4 ttl=62 time=0.098 ms
64 bytes from 10.2.0.252: icmp_seq=5 ttl=62 time=0.092 ms
64 bytes from 10.2.0.252: icmp_seq=6 ttl=62 time=0.083 ms
64 bytes from 10.2.0.252: icmp_seq=7 ttl=62 time=0.136 ms
64 bytes from 10.2.0.252: icmp_seq=8 ttl=62 time=0.097 ms
64 bytes from 10.2.0.252: icmp_seq=9 ttl=62 time=0.111 ms
64 bytes from 10.2.0.252: icmp_seq=10 ttl=62 time=0.085 ms
^C
--- 10.2.0.252 ping statistics ---
10 packets transmitted, 10 received, 0% packet loss, time 9156ms
rtt min/avg/max/mdev = 0.083/0.644/2.745/0.999 ms
mininet>
```

By pinging it can be seen that the time to ping the first packet has reduced from 4.5 milliseconds to 2 milliseconds.

(d)

We have made a change in the routing path between d1 and d6. This change is visible in the routing tables of r1 and r2 routers in the below two images.

For part (a)

```
mininet> r1 route -n
Kernel IP routing table
Destination     Gateway         Genmask         Flags Metric Ref    Use Iface
10.0.0.0        0.0.0.0         255.255.255.0   U     0      0        0 rs1
10.1.0.0        10.100.0.2      255.255.255.0   UG    0      0        0 r12
10.2.0.0        10.101.0.4      255.255.255.0   UG    0      0        0 r13
10.100.0.0      0.0.0.0         255.255.255.0   U     0      0        0 r12
10.101.0.0      0.0.0.0         255.255.255.0   U     0      0        0 r13
mininet> r2 route -n
Kernel IP routing table
Destination     Gateway         Genmask         Flags Metric Ref    Use Iface
10.0.0.0        10.100.0.1      255.255.255.0   UG    0      0        0 r21
10.1.0.0        0.0.0.0         255.255.255.0   U     0      0        0 rs2
10.2.0.0        10.102.0.6      255.255.255.0   UG    0      0        0 r23
10.100.0.0      0.0.0.0         255.255.255.0   U     0      0        0 r21
10.102.0.0      0.0.0.0         255.255.255.0   U     0      0        0 r23
mininet> r3 route -n
Kernel IP routing table
Destination     Gateway         Genmask         Flags Metric Ref    Use Iface
10.0.0.0        10.101.0.3      255.255.255.0   UG    0      0        0 r31
10.1.0.0        10.102.0.5      255.255.255.0   UG    0      0        0 r32
10.2.0.0        0.0.0.0         255.255.255.0   U     0      0        0 rs3
10.101.0.0      0.0.0.0         255.255.255.0   U     0      0        0 r31
10.102.0.0      0.0.0.0         255.255.255.0   U     0      0        0 r32
mininet> pingall
```

For part (c)

```
mininet> r1 route -n
Kernel IP routing table
Destination     Gateway         Genmask         Flags Metric Ref    Use Iface
10.0.0.0        0.0.0.0         255.255.255.0   U     0      0        0 rs1
10.1.0.0        10.100.0.2      255.255.255.0   UG    0      0        0 r12
10.2.0.0        10.100.0.2      255.255.255.0   UG    0      0        0 r12
10.100.0.0      0.0.0.0         255.255.255.0   U     0      0        0 r12
10.101.0.0      0.0.0.0         255.255.255.0   U     0      0        0 r13
mininet> r2 route -n
Kernel IP routing table
Destination     Gateway         Genmask         Flags Metric Ref    Use Iface
10.0.0.0        10.100.0.1      255.255.255.0   UG    0      0        0 r21
10.1.0.0        0.0.0.0         255.255.255.0   U     0      0        0 rs2
10.2.0.0        10.102.0.6      255.255.255.0   UG    0      0        0 r23
10.100.0.0      0.0.0.0         255.255.255.0   U     0      0        0 r21
10.102.0.0      0.0.0.0         255.255.255.0   U     0      0        0 r23
mininet> r3 rouote -n
bash: rouote: command not found
mininet> r3 route -n
Kernel IP routing table
Destination     Gateway         Genmask         Flags Metric Ref    Use Iface
10.0.0.0        10.102.0.5      255.255.255.0   UG    0      0        0 r32
10.1.0.0        10.102.0.5      255.255.255.0   UG    0      0        0 r32
10.2.0.0        0.0.0.0         255.255.255.0   U     0      0        0 rs3
10.101.0.0      0.0.0.0         255.255.255.0   U     0      0        0 r31
10.102.0.0      0.0.0.0         255.255.255.0   U     0      0        0 r32
mininet>
```

Question 2

(a)

- I opened Mininet in Kali Linux using these commands: "openvswitch-switch start" and "ls -1 /var/run/openswitch".Then I ran the python file containing the above topology "sudo mn --custom router.py --topo custom".

```
┌──(dchops㉿kali)-[~/Desktop]
└─$ sudo mn --custom q2.py --topo custom
*** No default OpenFlow controller found for default switch!
*** Falling back to OVS Bridge
*** Creating network
*** Adding controller
*** Adding hosts:
h1 h2 h3 h4
*** Adding switches:
s1 s2
*** Adding links:
(h1, s1) (h2, s1) (h3, s2) (h4, s2) (s1, s2)
*** Configuring hosts
h1 h2 h3 h4
*** Starting controller

*** Starting 2 switches
s1 s2 ...
*** Starting CLI:
mininet>
```

Fig 2.1

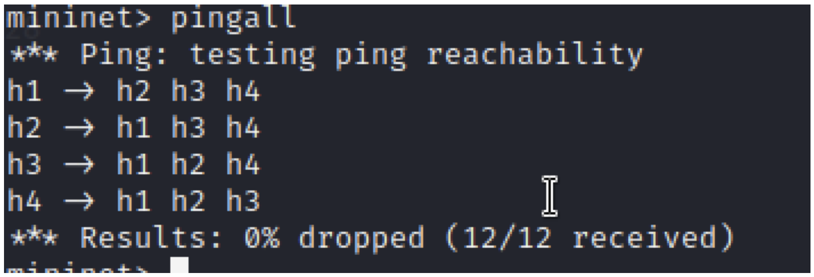


Fig 2.2: On doing "pingall" we can see all hosts can send packets to each other.

- Now I implemented a simple TCP client-server system with the ability to perform file operations and directory navigation. The communication between the client and server can be encrypted using different modes (plaintext, substitution cipher, and transpose cipher). I made 3 files: client.py, server.py and utilities.py.

TCP Client (client.py):

- Connection Setup: The client continuously connects to a specified server IP address and port. H4 in this case(10.0.0.4)
- User Interaction: The user selects an encryption mode (plaintext, substitution cipher, transpose cipher).
- Command Handling: The client sends commands to the server, including file download/upload (dwd/upd), changing the directory (cwd), listing directory contents (ls), and exiting (exit)
- Encryption and Decryption: Based on the selected encryption mode, the client encrypts messages before sending and decrypts received messages.

TCP Server (server.py):

- Connection Setup: The server listens on a specified IP address and port, waiting for incoming client connections.
- Client Handling: When a client connects, the server processes commands received from the client based on the selected encryption mode.
- Command Execution: Server executes commands such as changing the directory, listing directory contents, and handling file download/upload.
- Encryption and Decryption:Server uses encryption and decryption based on the selected encryption mode.

Shared Utilities (utilities.py):

Contains utility functions for sending and receiving messages, creating a listening socket, and handling the current directory.

Overall Flow:

- Client-Server Interaction: Clients connect to the server, select an encryption mode, and send commands.The server receives commands, executes them, and sends back responses.
- File Operations: Clients can download/upload files, and the server handles these operations.
- Directory Navigation: Commands like changing the directory and listing directory contents are supported.
- Encryption Options: Clients and servers can use different encryption modes for secure communication.

Parameters like congestion control algorithm could be added while running the iperf command. Configuration could be set up as required using “xterm”. Link loss is mentioned in the code itself of q2.py.

(b) On running the client on H1 and the server on H4 for different congestion control algorithms
→ BBR: Avg throughput = Transfer / interval
= 10.3Mbps

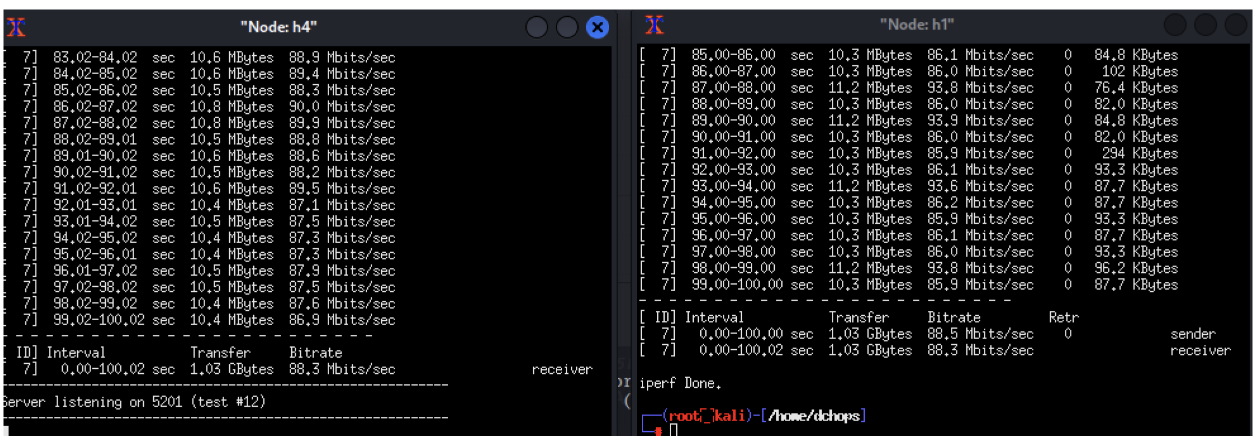


Fig 2.3

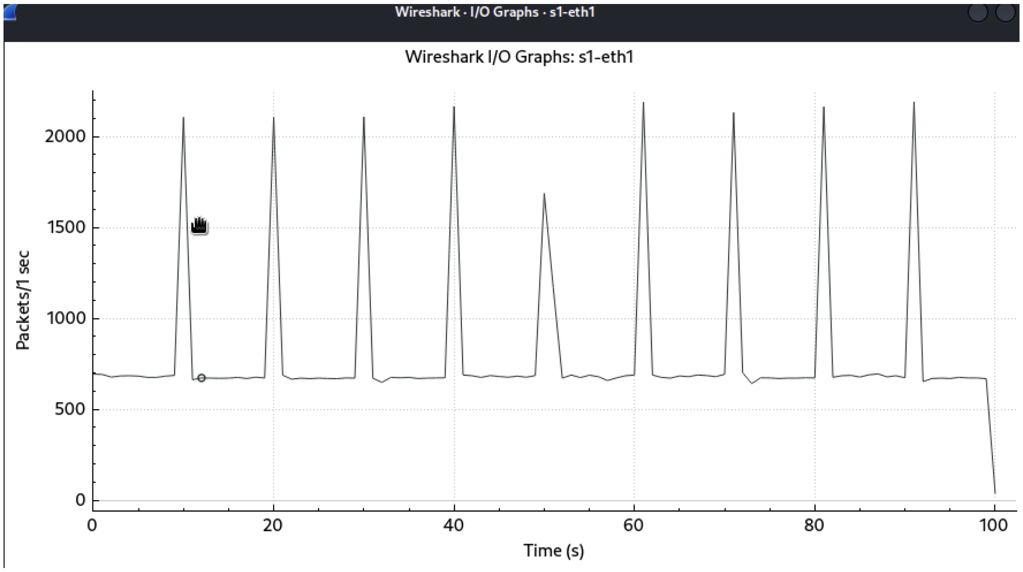


Fig 2.4

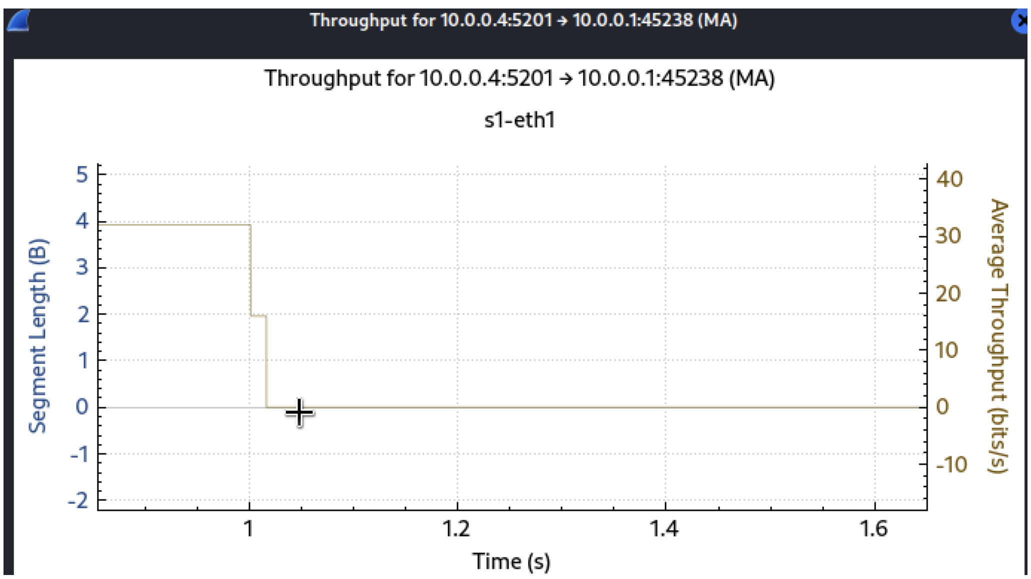


Fig 2.5

BBR focuses on probing for available bandwidth and adjusting its sending rate accordingly. It tries to utilize the available bandwidth efficiently without causing congestion. The graph might show a more consistent throughput compared to Cubic.

→ Cubic, Avg throughput =Transfer / interval
=9.94Mbps

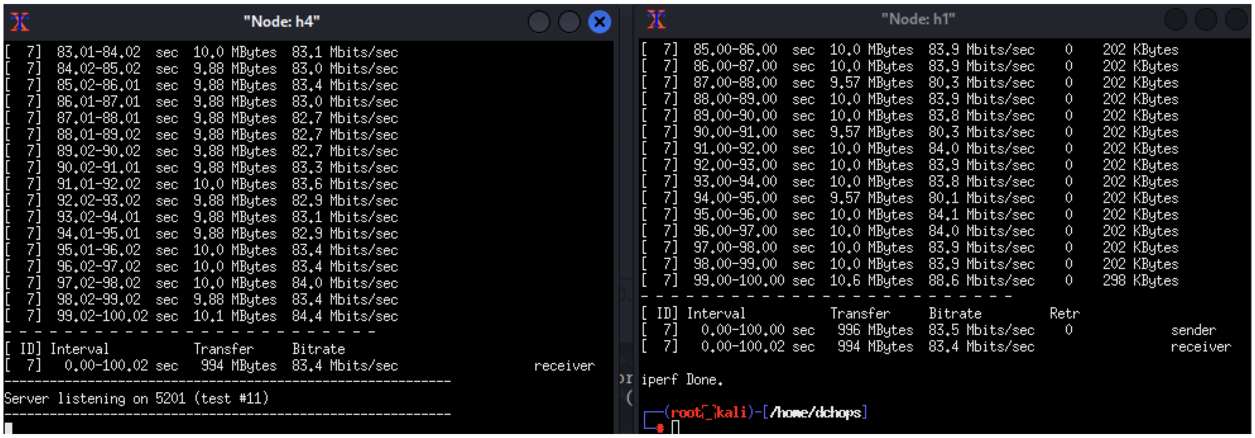

"Node: h4"
[7] 83.01-84.02 sec 10.0 MBytes 83.1 Mbits/sec
[7] 84.02-85.02 sec 9.88 MBytes 83.0 Mbits/sec
[7] 85.02-86.01 sec 9.88 MBytes 83.4 Mbits/sec
[7] 86.01-87.01 sec 9.88 MBytes 83.0 Mbits/sec
[7] 87.01-88.01 sec 9.88 MBytes 82.7 Mbits/sec
[7] 88.01-89.02 sec 9.88 MBytes 82.7 Mbits/sec
[7] 89.02-90.02 sec 9.88 MBytes 82.7 Mbits/sec
[7] 90.02-91.01 sec 9.88 MBytes 83.3 Mbits/sec
[7] 91.01-92.02 sec 10.0 MBytes 83.6 Mbits/sec
[7] 92.02-93.02 sec 9.88 MBytes 82.9 Mbits/sec
[7] 93.02-94.01 sec 9.88 MBytes 83.1 Mbits/sec
[7] 94.01-95.01 sec 9.88 MBytes 82.9 Mbits/sec
[7] 95.01-96.02 sec 10.0 MBytes 83.4 Mbits/sec
[7] 96.02-97.02 sec 10.0 MBytes 83.4 Mbits/sec
[7] 97.02-98.02 sec 10.0 MBytes 84.0 Mbits/sec
[7] 98.02-99.02 sec 9.88 MBytes 83.4 Mbits/sec
[7] 99.02-100.02 sec 10.1 MBytes 84.4 Mbits/sec
[ID] Interval   Transfer   Bitrate
[7]   0.00-100.02 sec   994 MBytes 83.4 Mbits/sec   receiver
Server listening on 5201 (test #11)
"Node: h1"
[7] 85.00-86.00 sec 10.0 MBytes 83.9 Mbits/sec   0   202 KBytes
[7] 86.00-87.00 sec 10.0 MBytes 83.9 Mbits/sec   0   202 KBytes
[7] 87.00-88.00 sec 9.57 MBytes 80.3 Mbits/sec   0   202 KBytes
[7] 88.00-89.00 sec 10.0 MBytes 83.9 Mbits/sec   0   202 KBytes
[7] 89.00-90.00 sec 10.0 MBytes 83.8 Mbits/sec   0   202 KBytes
[7] 90.00-91.00 sec 9.57 MBytes 80.3 Mbits/sec   0   202 KBytes
[7] 91.00-92.00 sec 10.0 MBytes 84.0 Mbits/sec   0   202 KBytes
[7] 92.00-93.00 sec 10.0 MBytes 83.9 Mbits/sec   0   202 KBytes
[7] 93.00-94.00 sec 10.0 MBytes 83.8 Mbits/sec   0   202 KBytes
[7] 94.00-95.00 sec 9.57 MBytes 80.1 Mbits/sec   0   202 KBytes
[7] 95.00-96.00 sec 10.0 MBytes 84.1 Mbits/sec   0   202 KBytes
[7] 96.00-97.00 sec 10.0 MBytes 84.0 Mbits/sec   0   202 KBytes
[7] 97.00-98.00 sec 10.0 MBytes 83.9 Mbits/sec   0   202 KBytes
[7] 98.00-99.00 sec 10.0 MBytes 83.9 Mbits/sec   0   202 KBytes
[7] 99.00-100.00 sec 10.6 MBytes 88.6 Mbits/sec   0   298 KBytes
[ID] Interval   Transfer   Bitrate   Retr
[7]   0.00-100.00 sec   996 MBytes 83.5 Mbits/sec   0   sender
[7]   0.00-100.02 sec   994 MBytes 83.4 Mbits/sec   receiver
iperf Done.
(root㉿kali)-[/home/dchops]

Fig 2.6

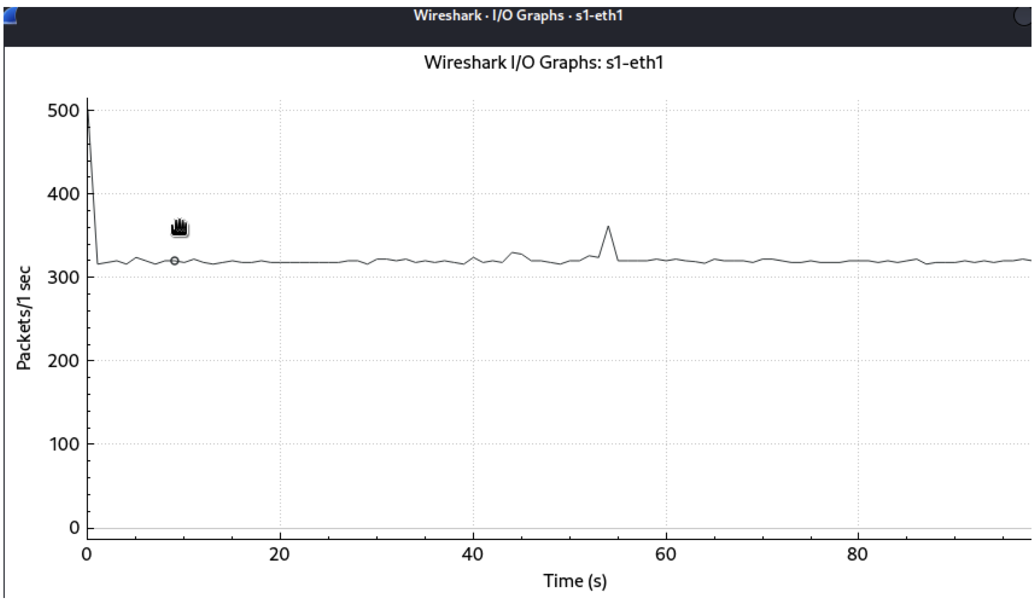

Wireshark · I/O Graphs · s1-eth1
Wireshark I/O Graphs: s1-eth1
Packets/1 sec
500
400
300
200
100
0
0
20
40
60
80
Time (s)

Fig 2.7

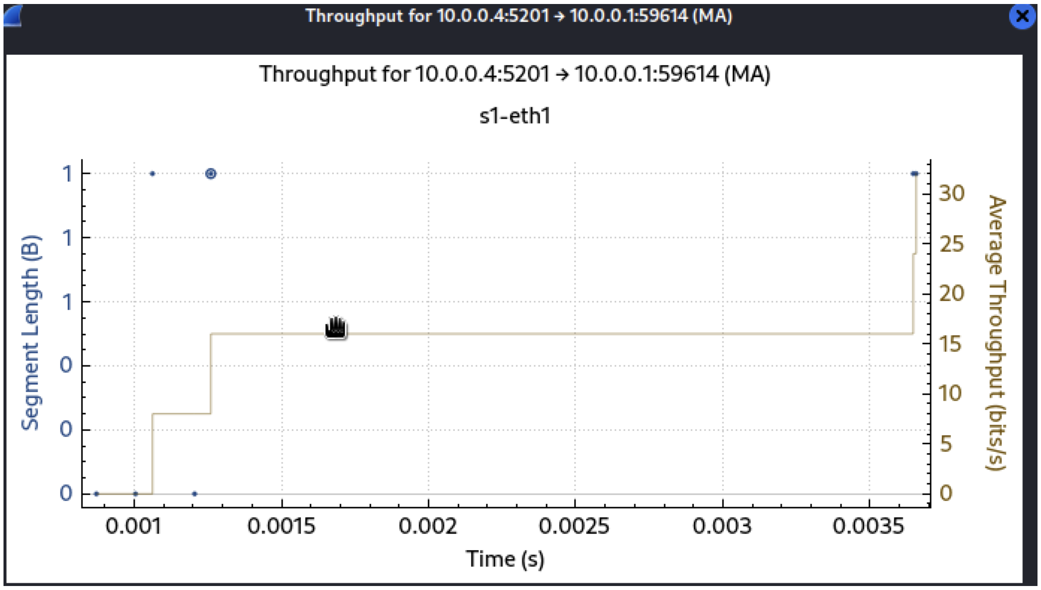


Fig 2.8

Cubic aggressively increases its sending rate until packet loss is detected. After that, it rapidly reduces the rate to avoid congestion. The graph might show bursts of high throughput followed by sharp drops.

→ Reno, Avg throughput = Transfer / Interval
=10.375Mbps

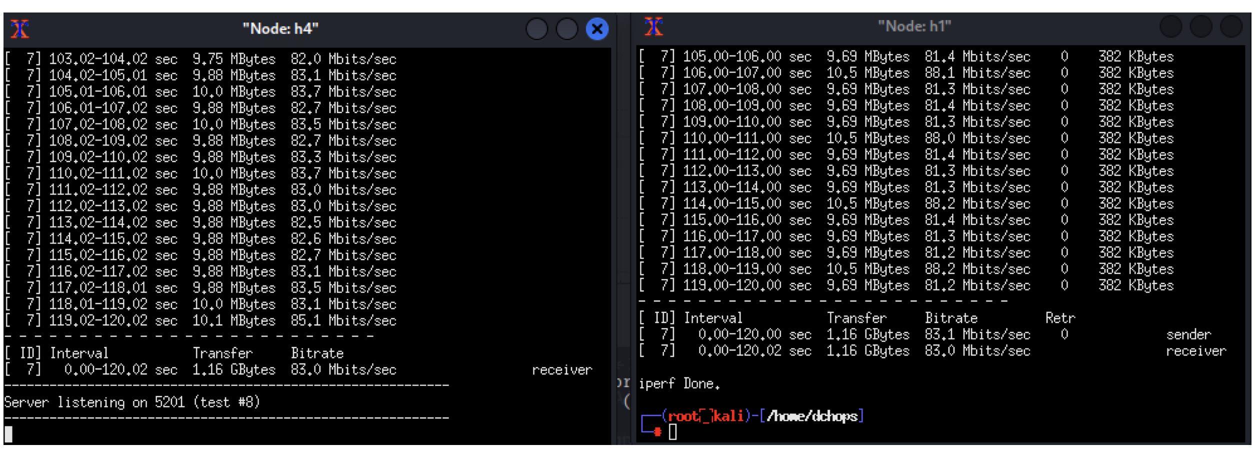


Fig 2.9

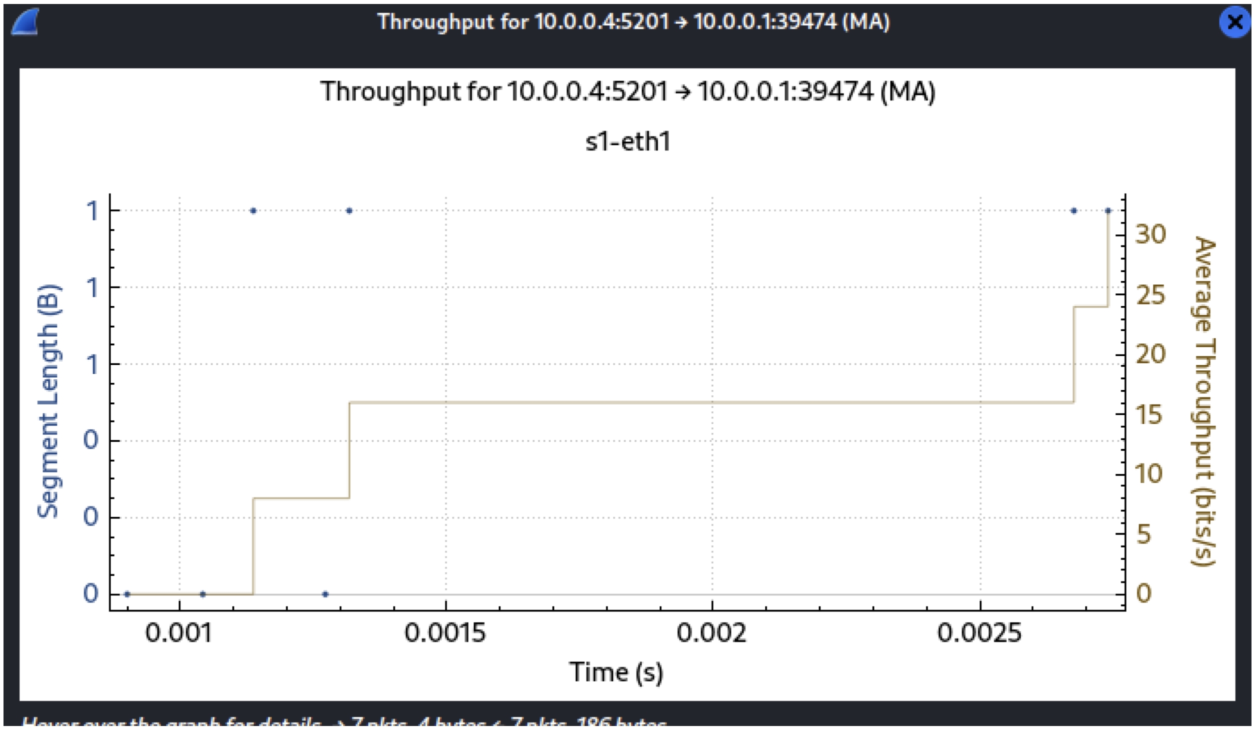


Fig 2.10

Reno is an older congestion control algorithm. It increases its sending rate slowly and sharply reduces it upon packet loss. The graph shows a more gradual increase in throughput compared to Cubic, with less aggressive rate changes.

→ Vegas, Avg throughput =Transfer/interval

=11.4Mbps

Fig 2.11

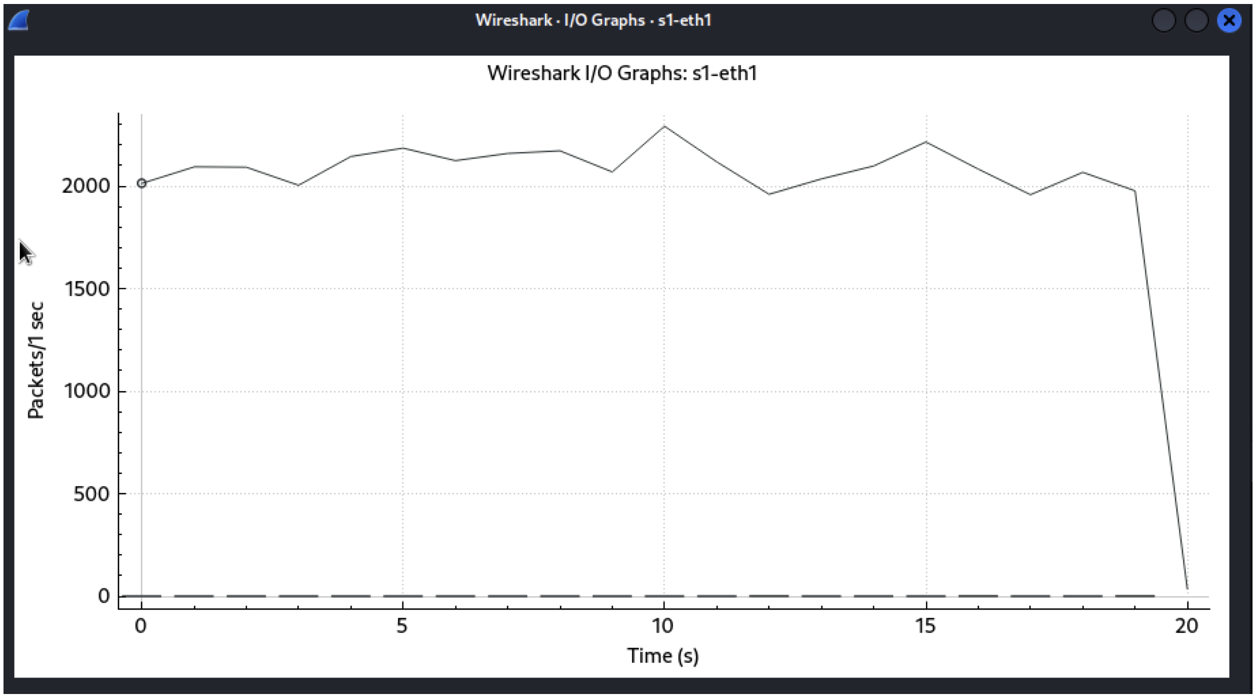


Fig 2.12

Vegas uses an estimate of the round-trip time and aims to keep the network operating at the point where the delay begins to increase due to congestion. The graph shows a steady increase in throughput with fewer abrupt drops.

(c) On running the client on H1, H2, H3 simultaneously and the server on H4.

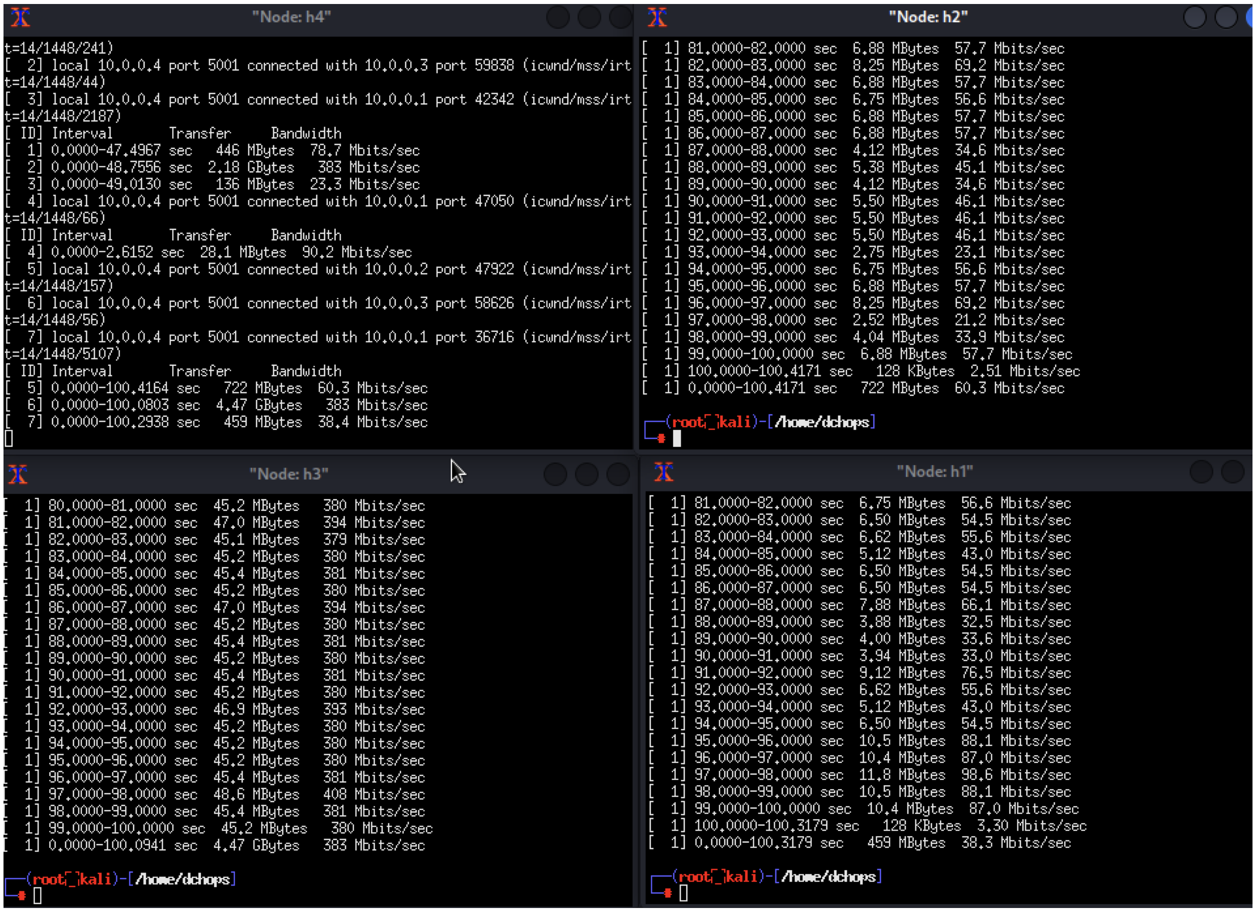


Fig 2.13

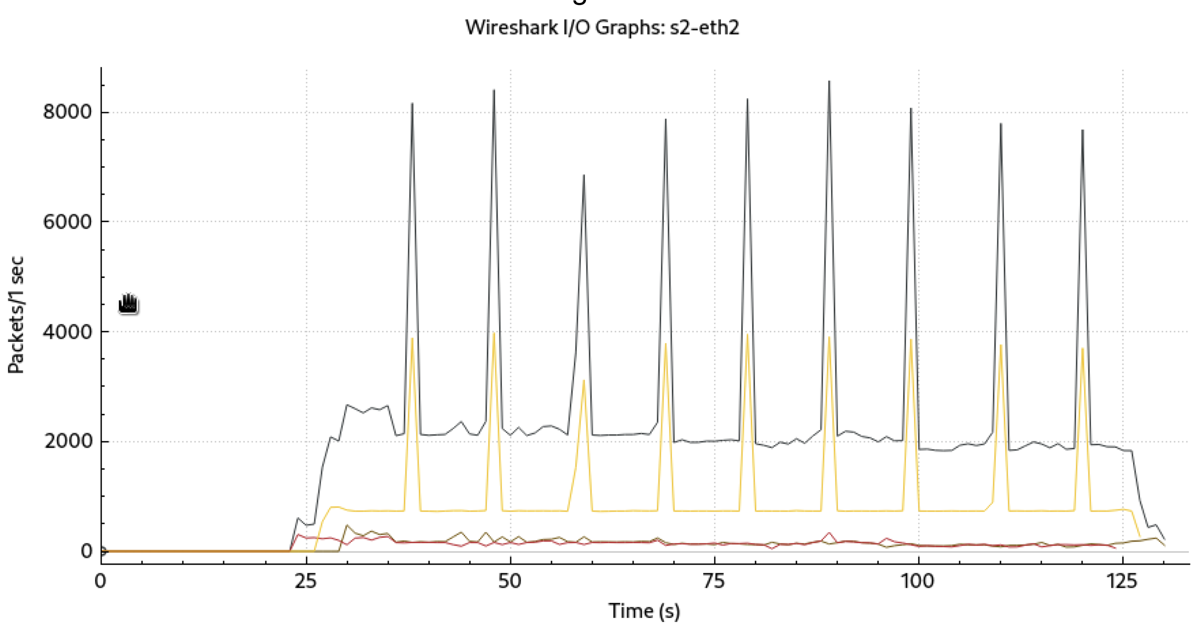


Fig 2.14

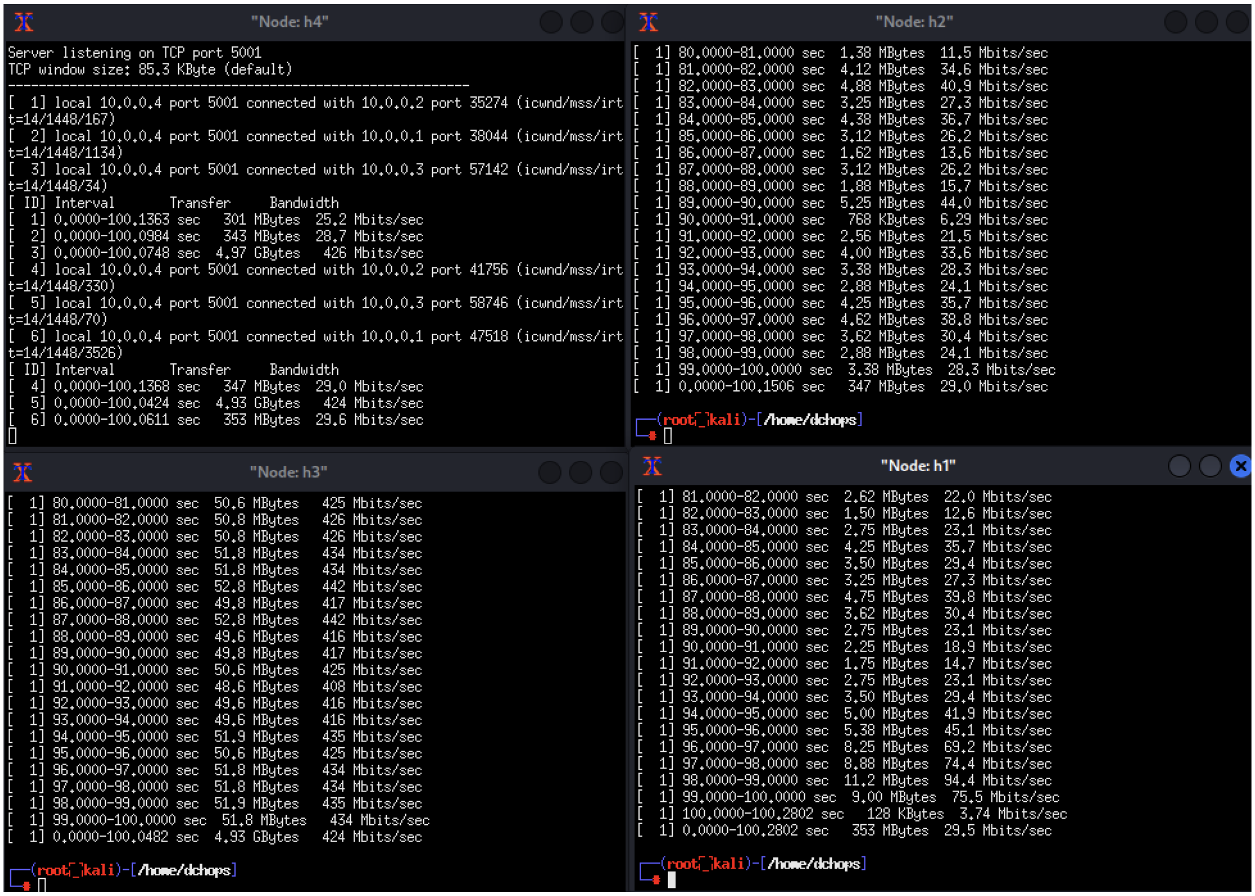


Fig 2.15

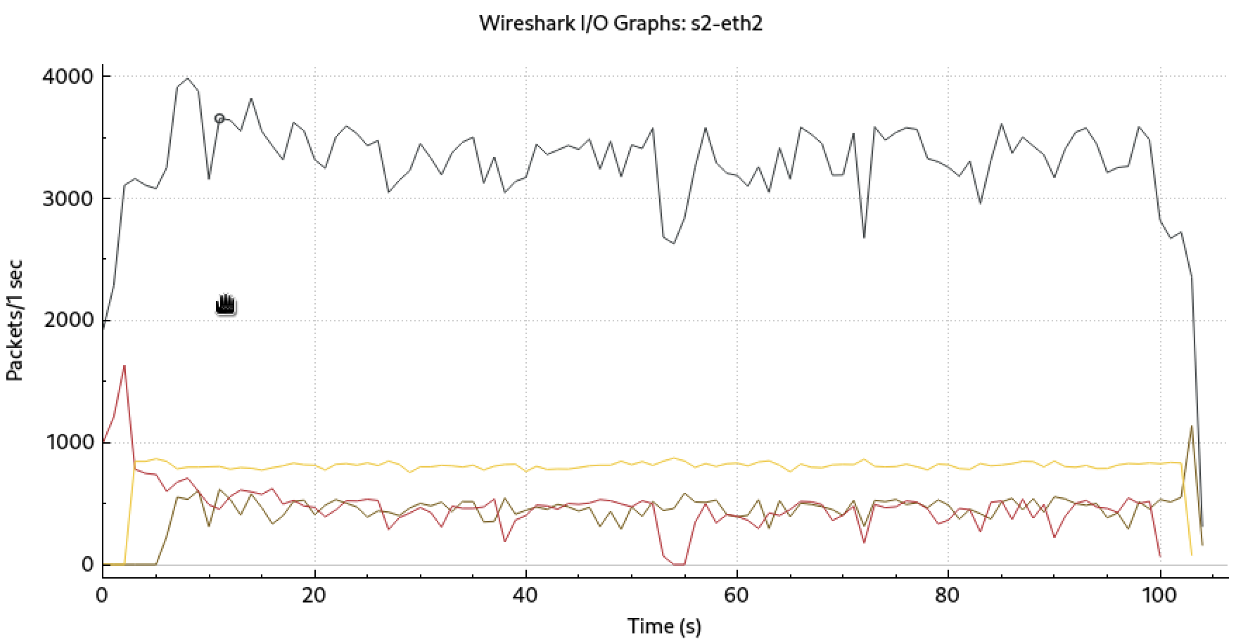


Fig 2.16

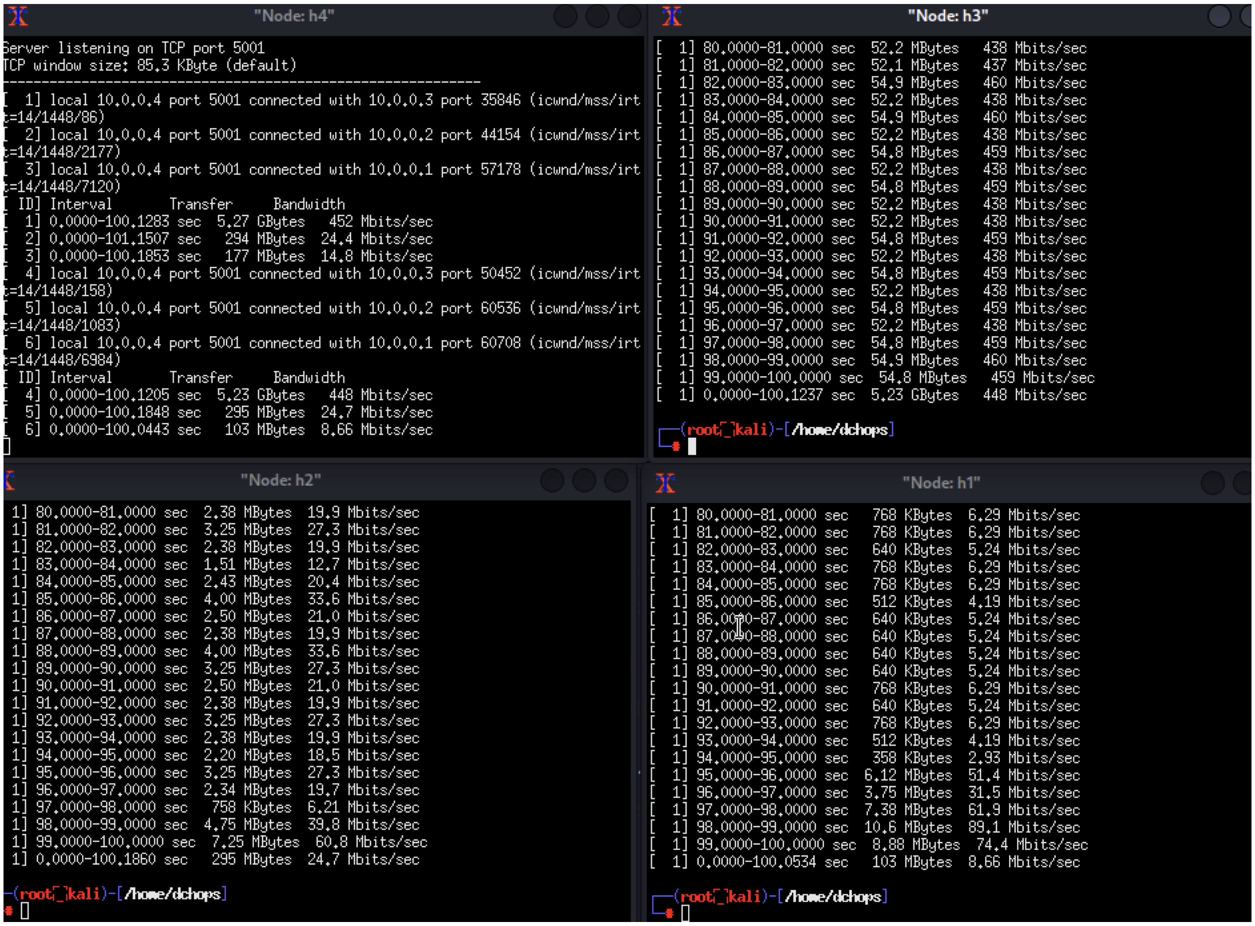


Fig 2.17

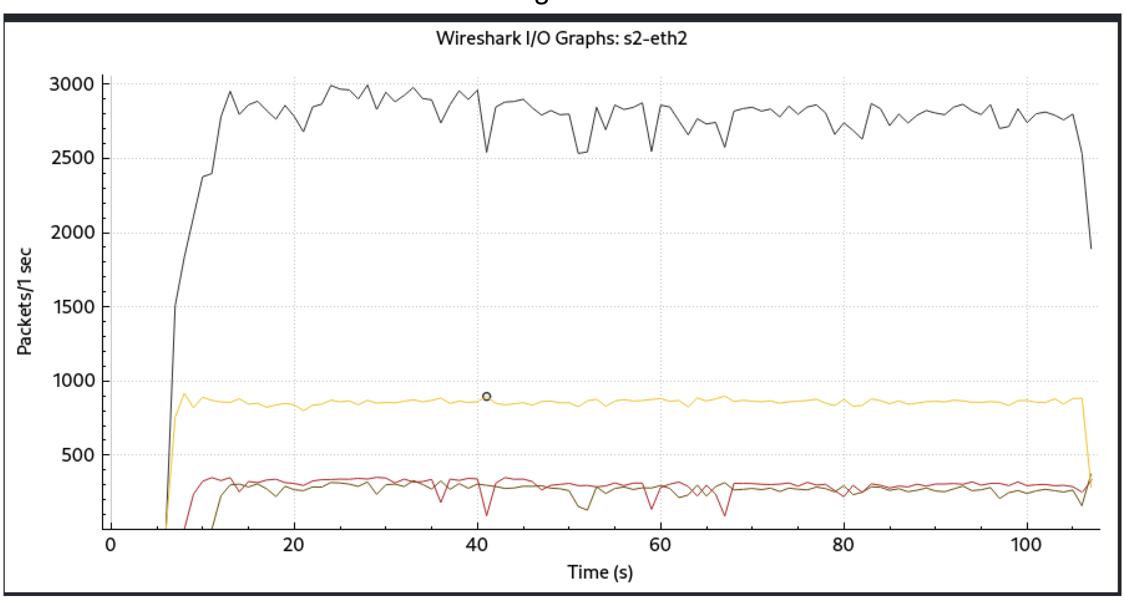


Fig 2.18

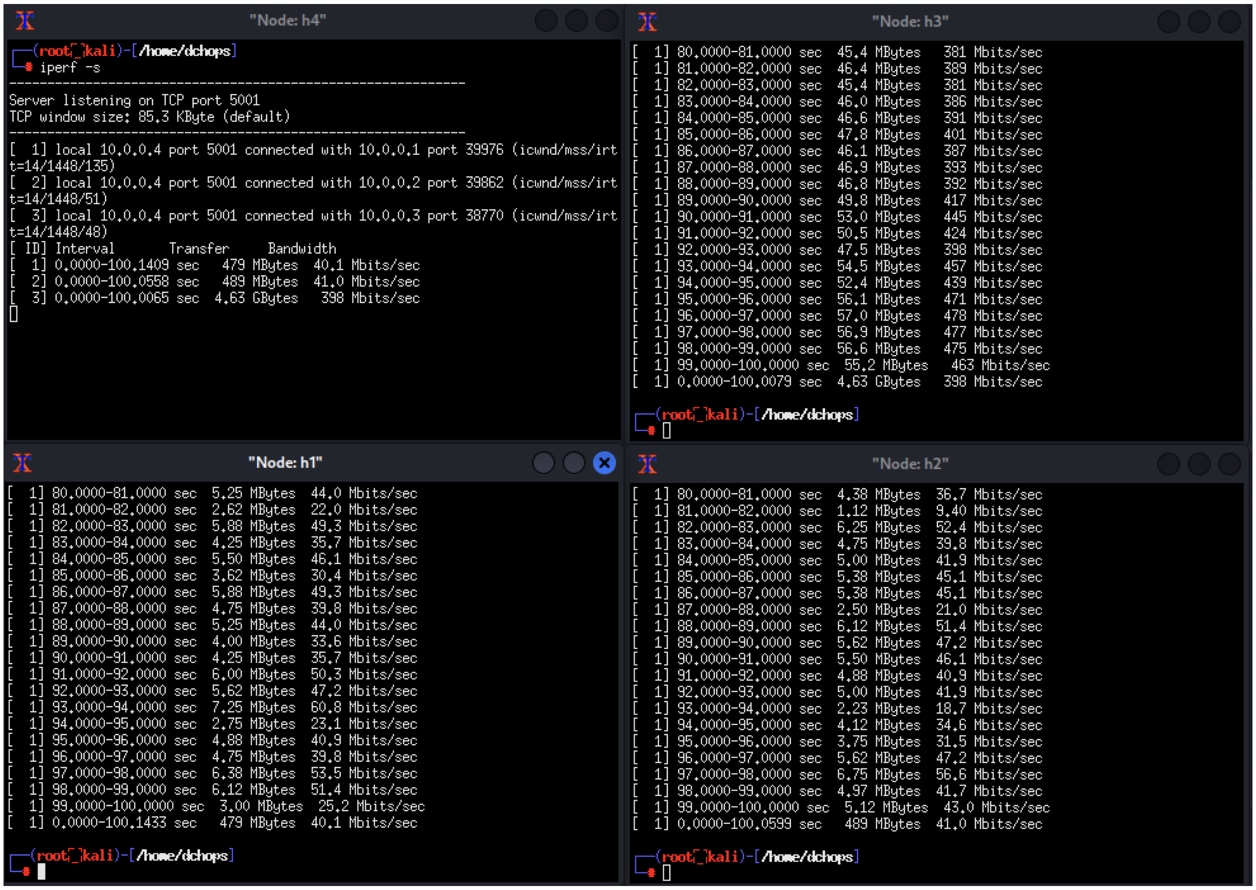


Fig 2.19

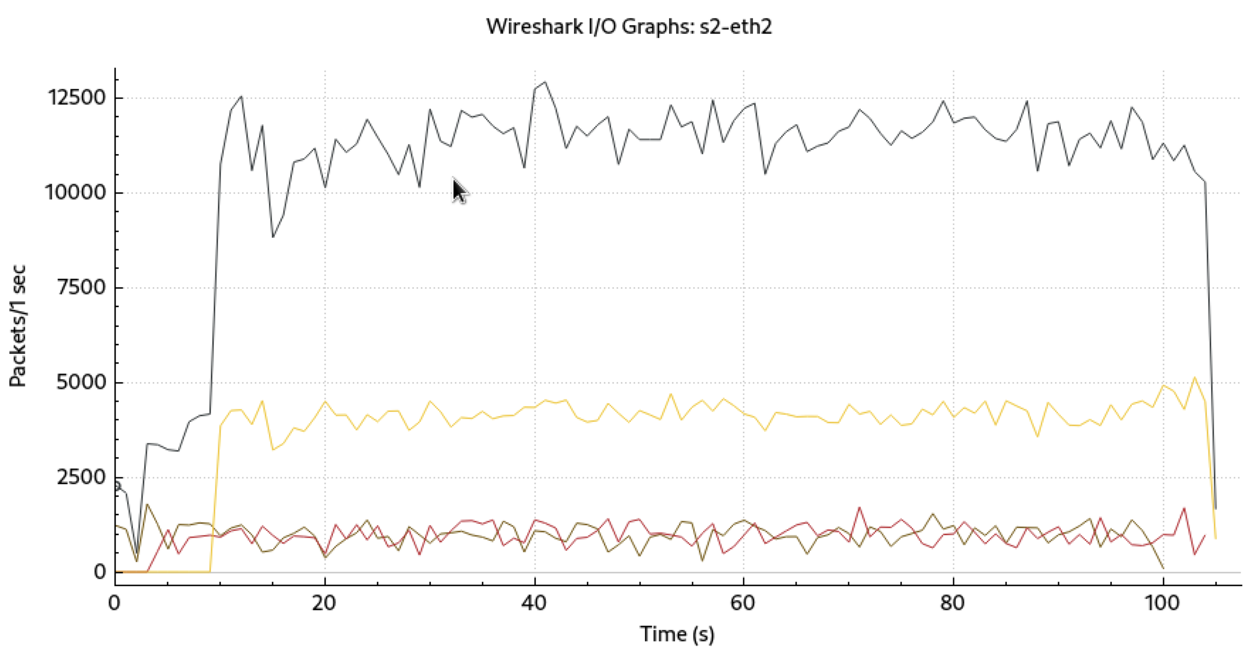


Fig 2.20

Different colors show no. of packets by a specific client. Yellow-h3 red-h2 and golden is h1.

As seen the avg throughput is decreased due to multiple clients connected to same server, for each congestion control scheme for h1 throughput is decreased from the previous part.

→ Throughput table

Congestion control type	h1	h2	h3
BBR	4.59	7.22	44.7
cubic	2.95	3.47	49.3
reno	1.03	2.95	52.3
vegas	4.79	1.00	46.3

(d) after configuring the tink loss parameter of the middle link (s1 - s2) to 1% and running the experiment in (b).

→ BBR, Avg throughput= 10.2Mbps

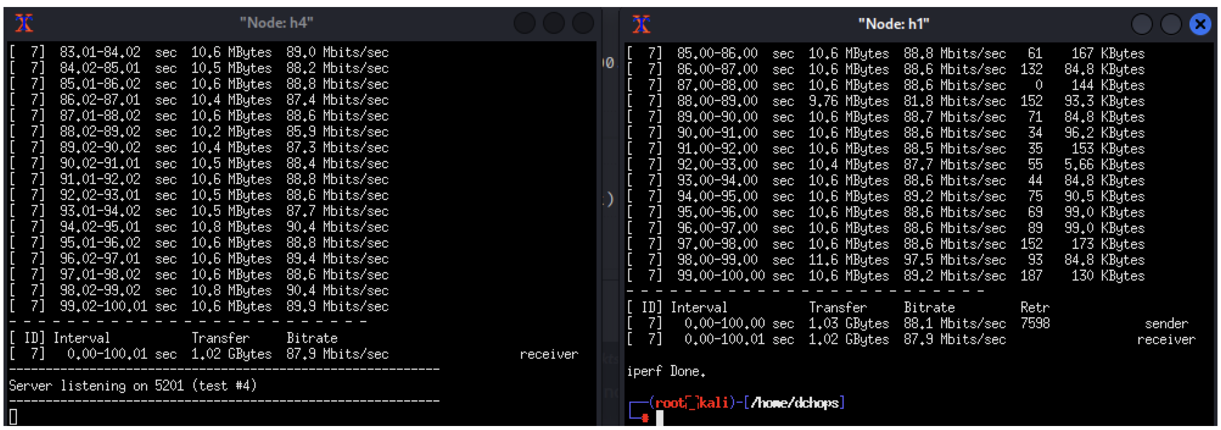


Fig 2.21

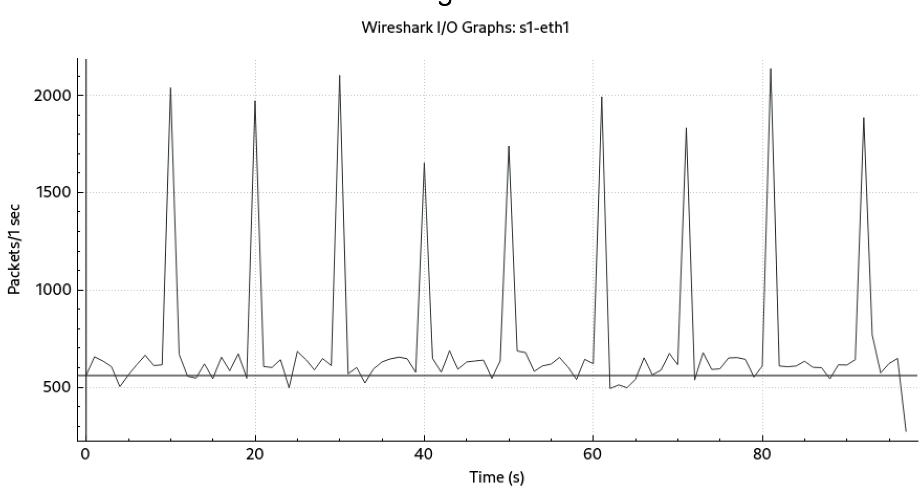


Fig 2.22

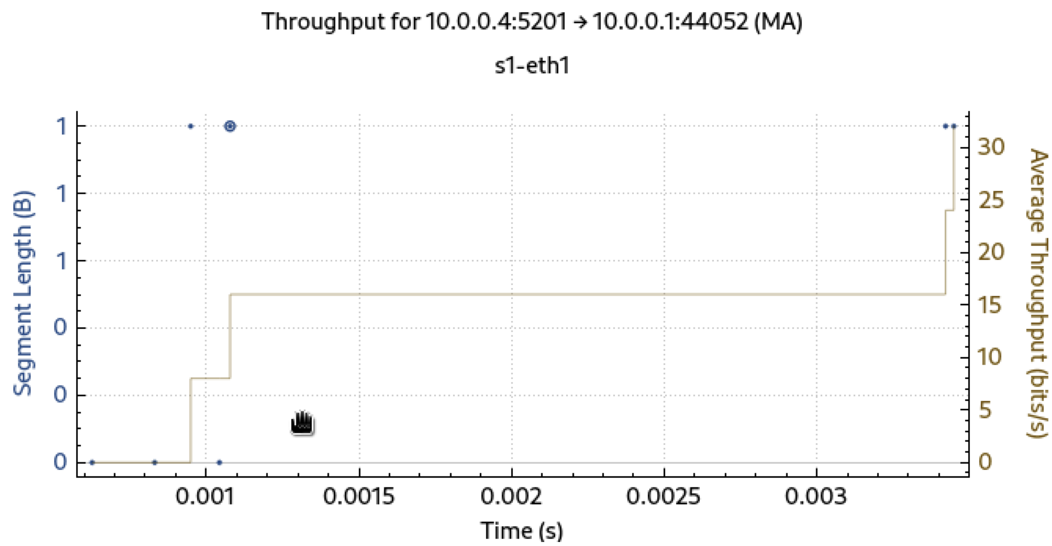


Fig 2.23

→ Cubic , Avg throughput=9.36Mbps

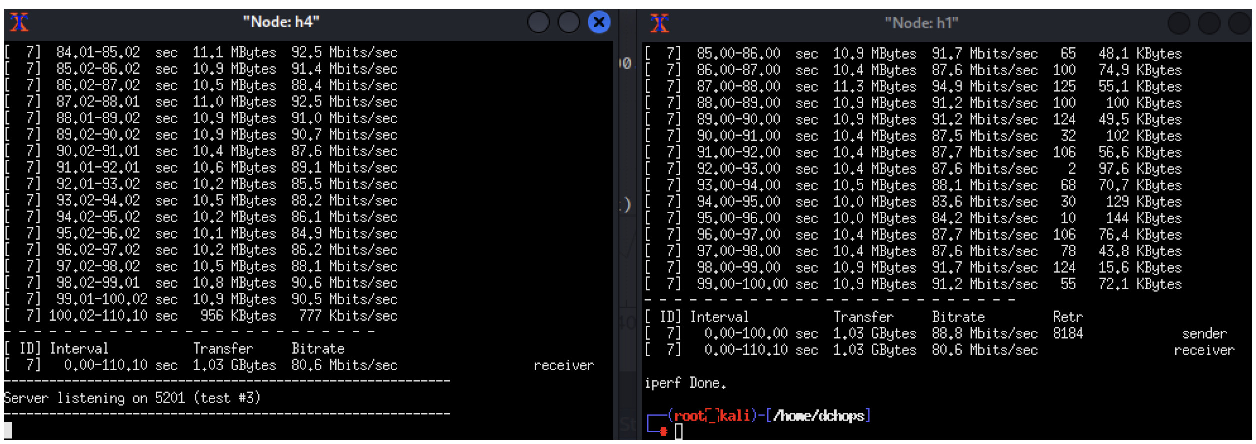


Fig 2.24

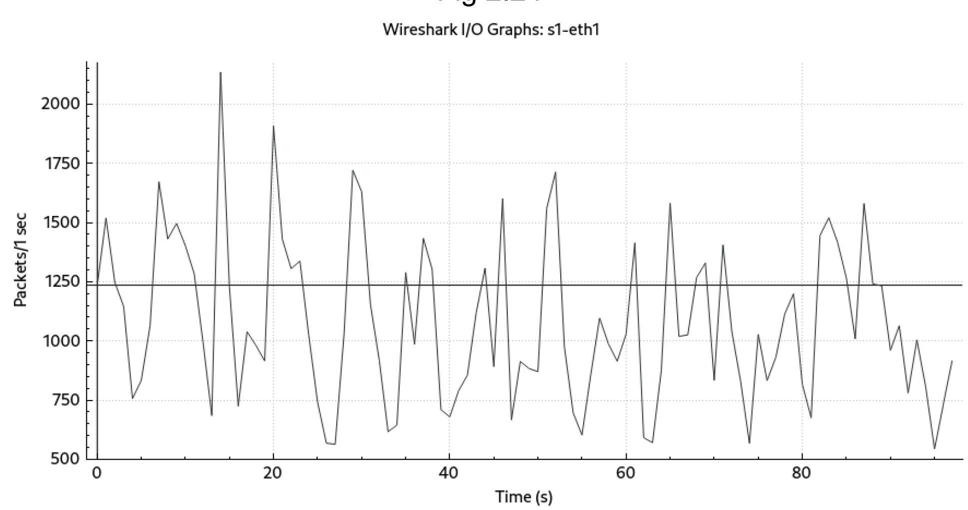


Fig 2.25

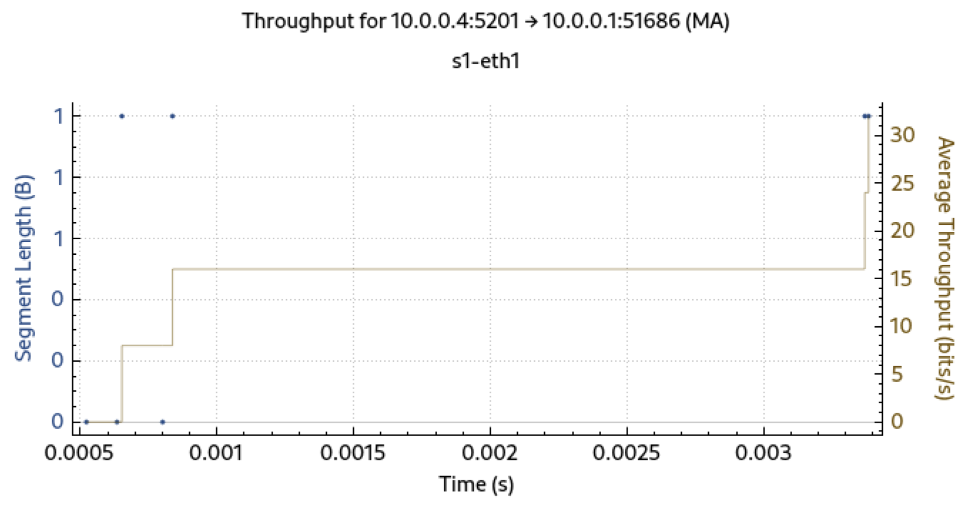


Fig 2.26

→ Reno, Avg throughput=10.2Mbps

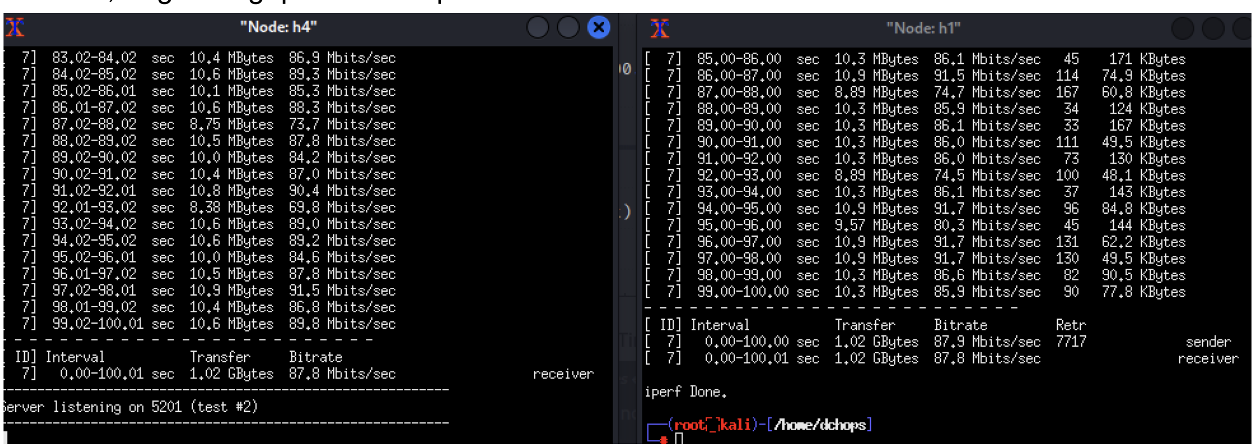


Fig 2.27

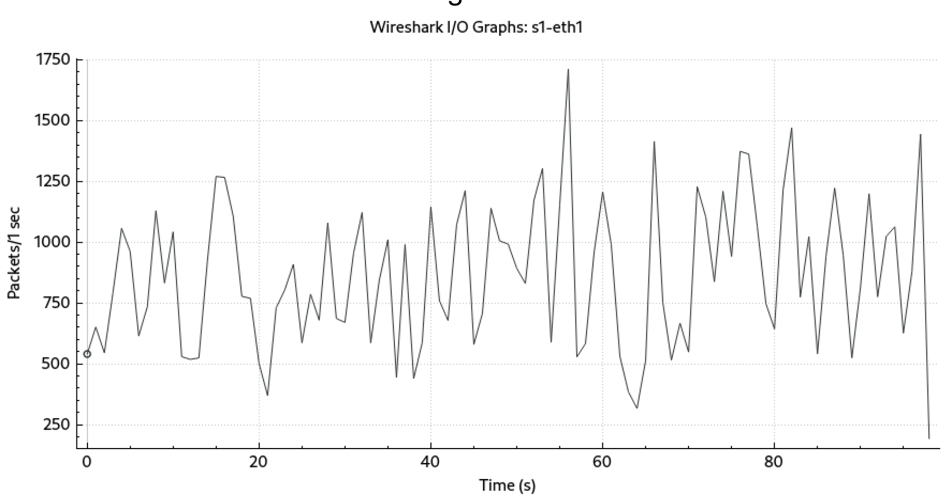


Fig 2.28

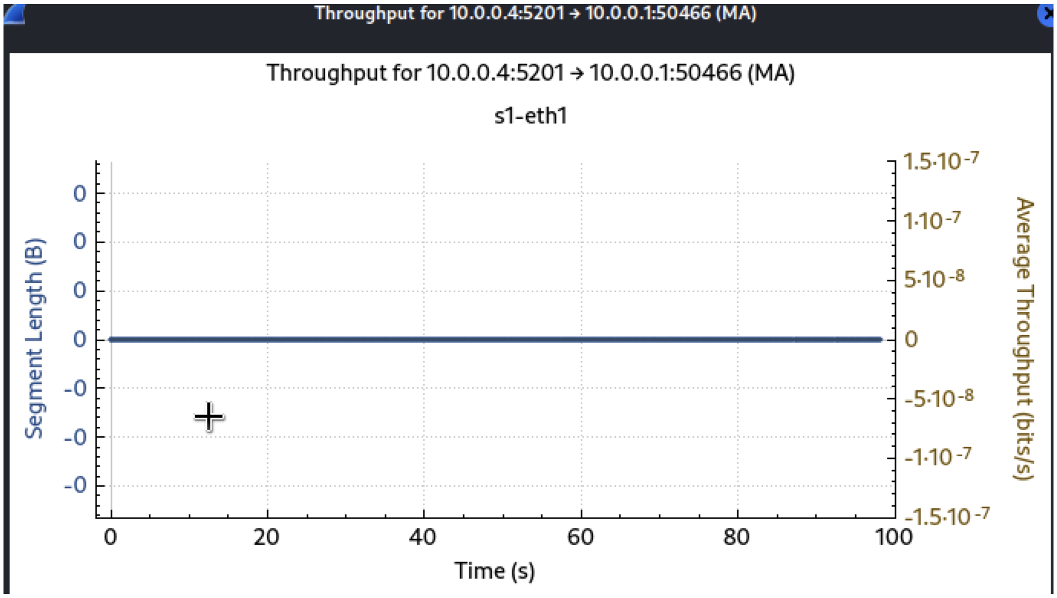


Fig 2.29

→ Vegas, Avg throughput=10.7Gbps

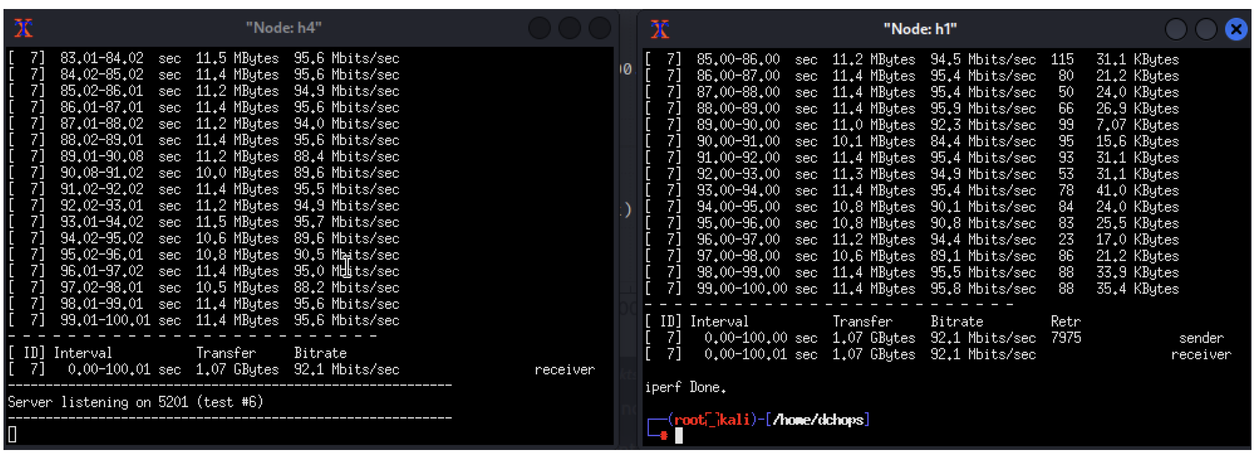


Fig 2.30

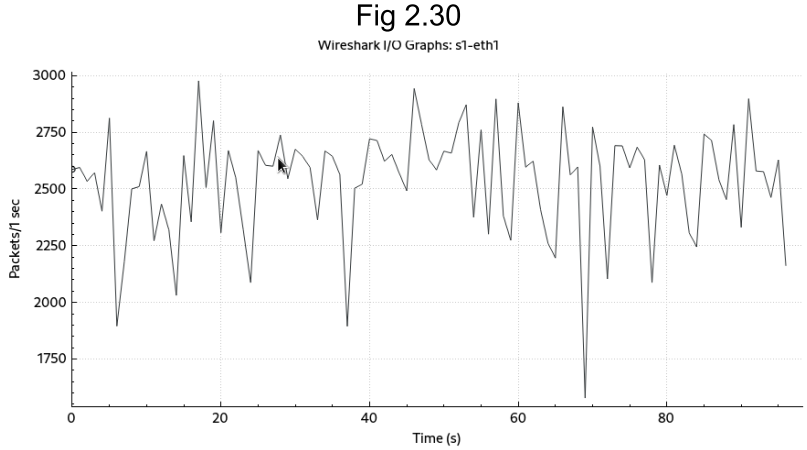


Fig 2.31

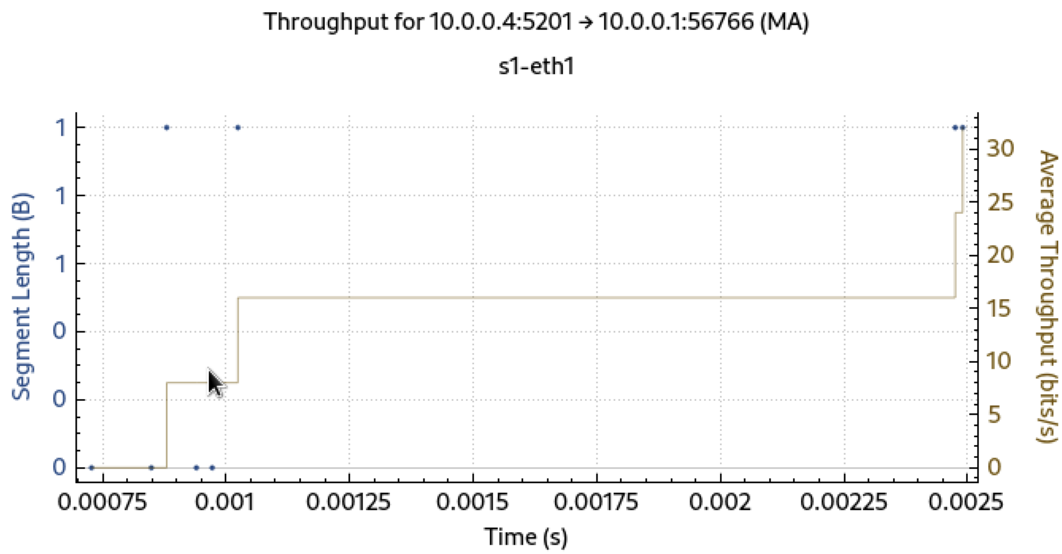


Fig 2.32

After configuring the tink loss parameter of the middle link (s1 - s2) to 3% and running the experiment in (b).

→ Bbr, Avg throughput=9.59Mbps

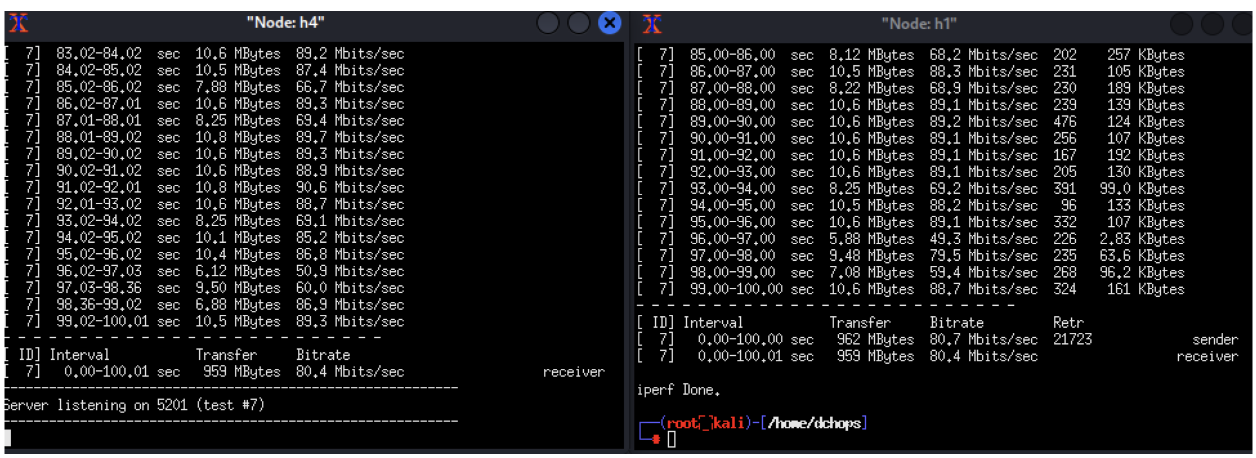


Fig 2.33

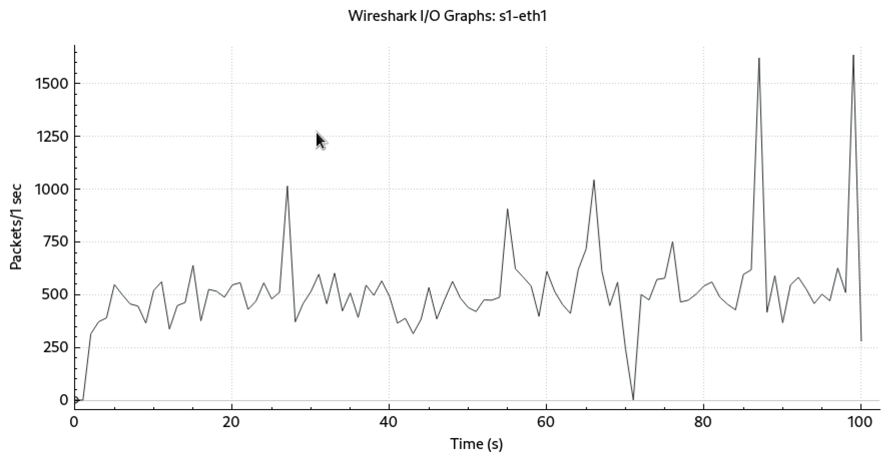


Fig 2.34

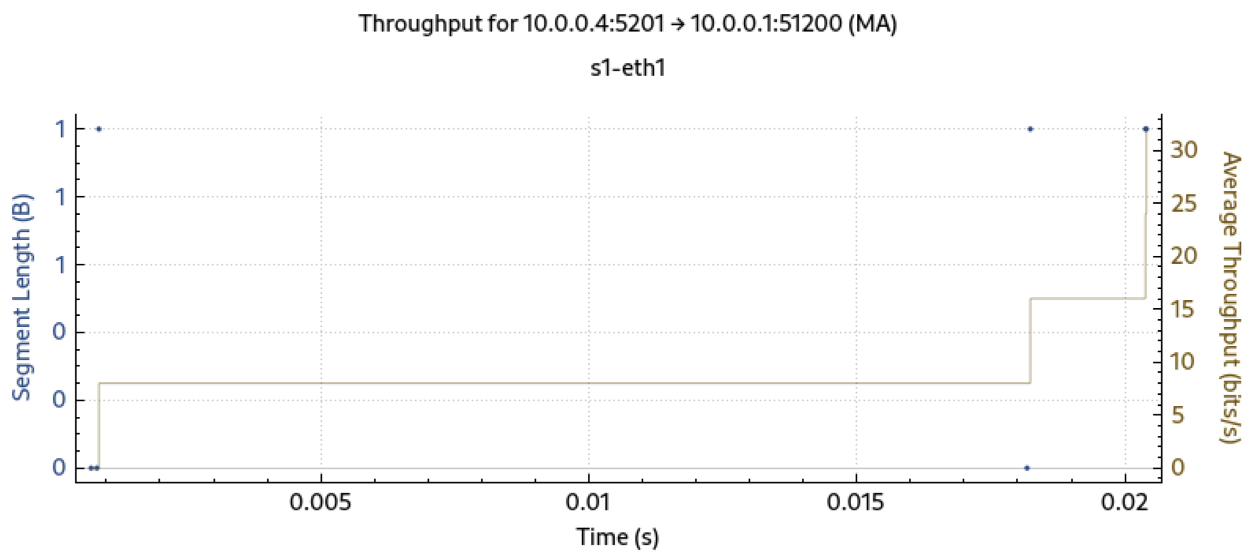


Fig 2.35

→ Cubic, Avg throughput=7.88Mbps

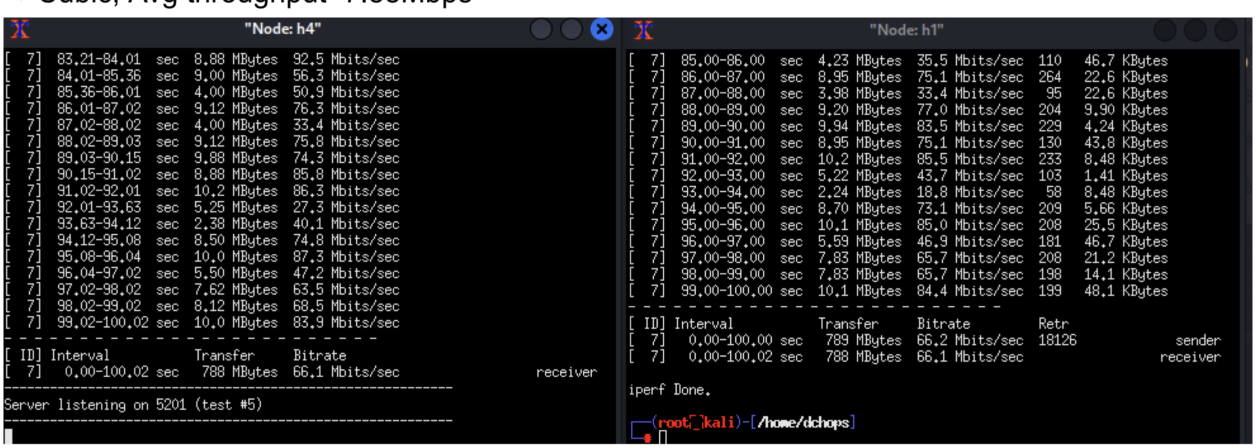


Fig 2.36

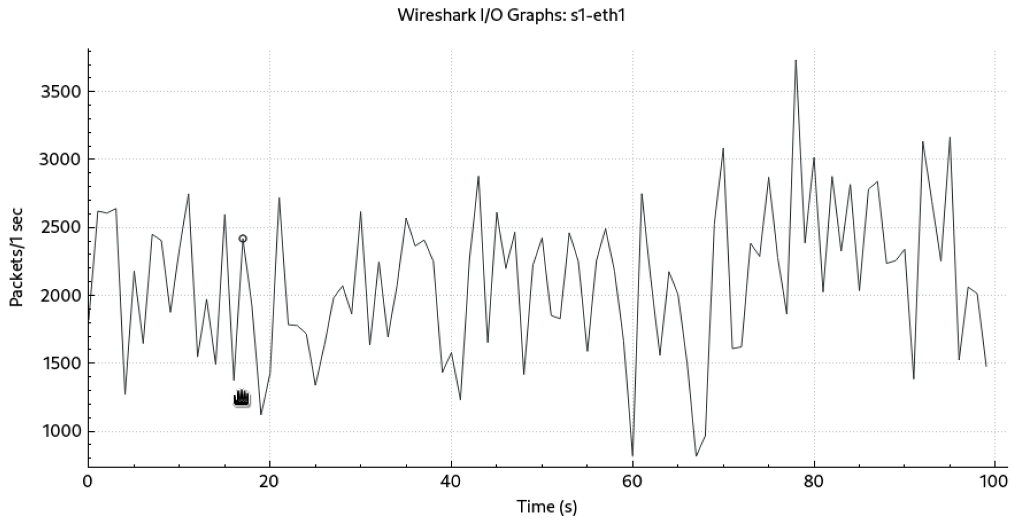


Fig 2.37

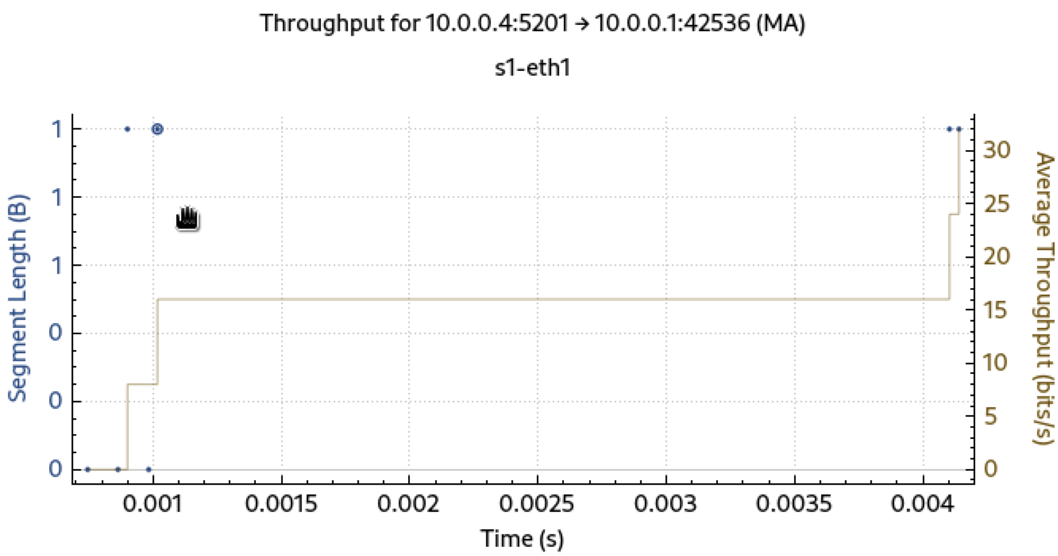


Fig 2.38

→ Reno, Avg throughput=9.40Mbps

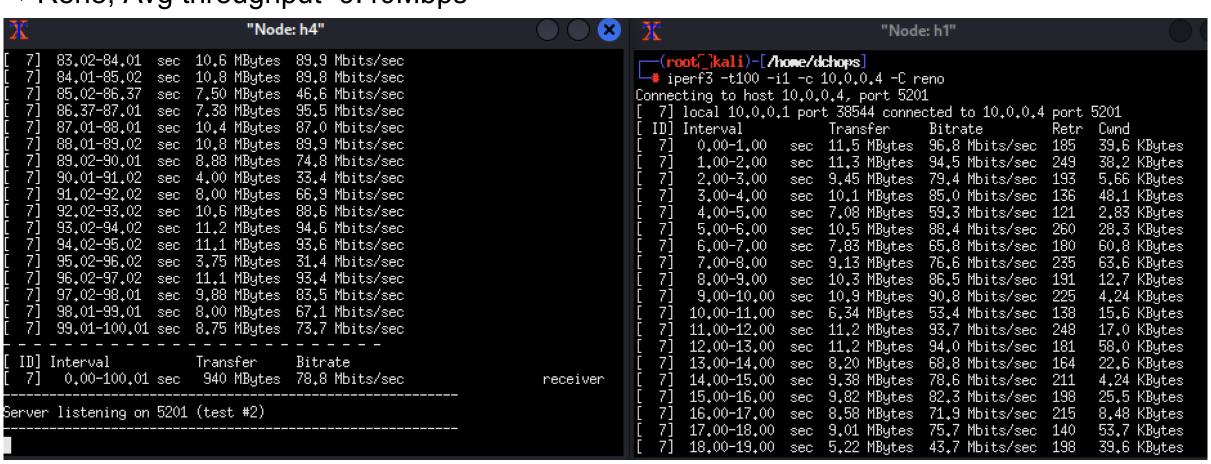


Fig 2.39

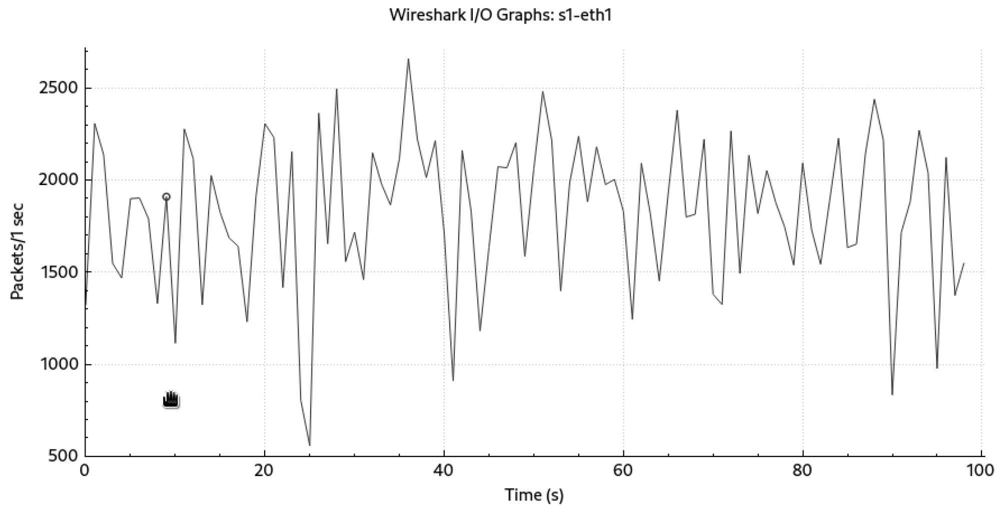


Fig 2.40

Throughput for 10.0.0.4:5201 → 10.0.0.1:38544 (MA)

s1-eth1

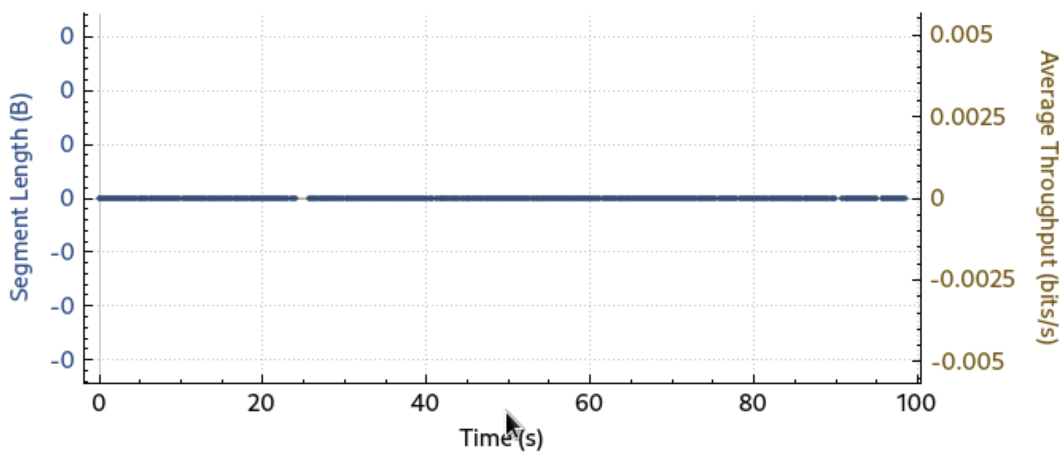


Fig 2.41

→ Vegas, Avg throughput=6.37Mbps

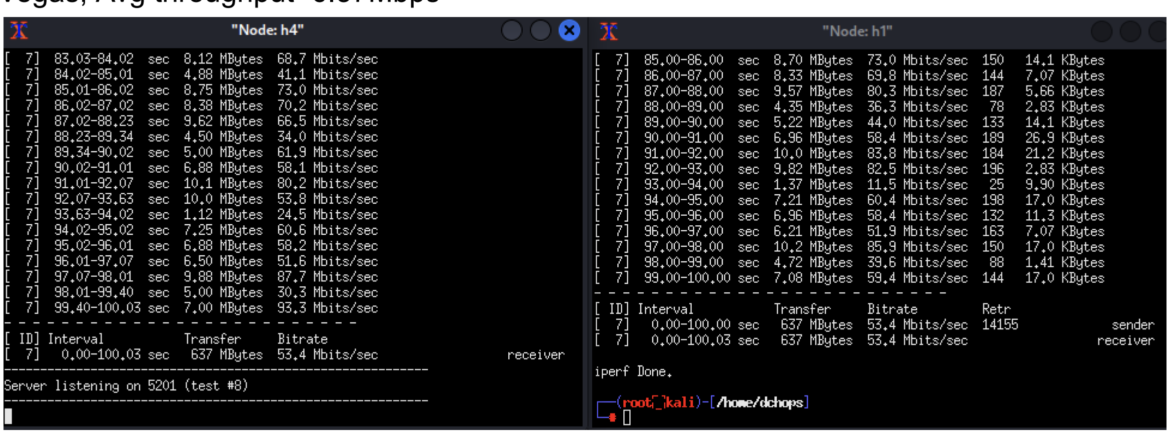


Fig 2.42

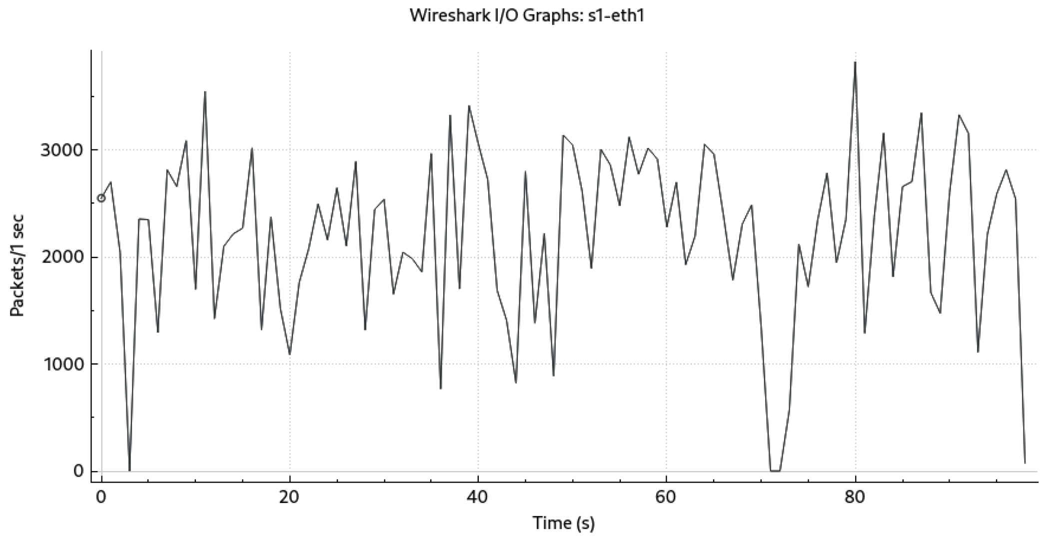


Fig 2.43

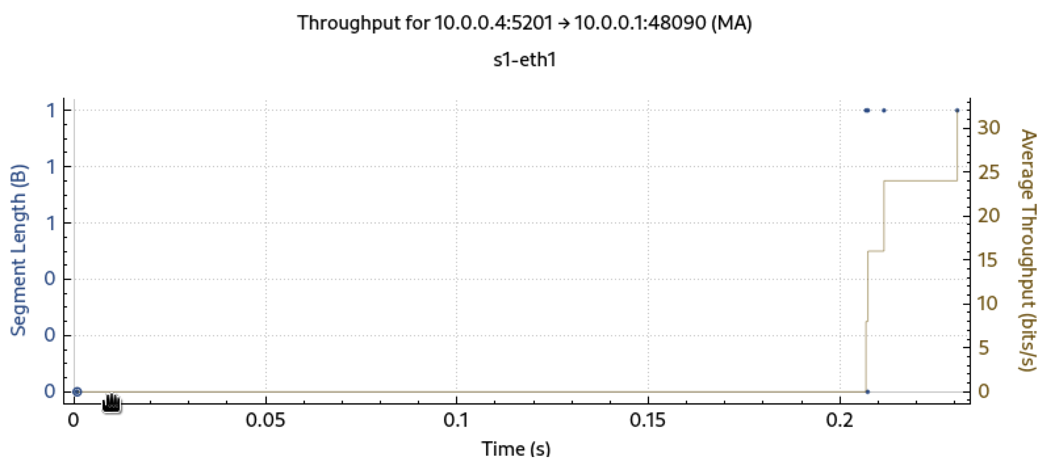


Fig 2.44

As the link loss parameter is increased from 1 to 3, avg throughput for each scheme is reduced. It is because if the loss increases, retransmissions increase, and therefore, the actual throughput decreases.