Assignment 2

Team Members:
Ayush Modi 21110039
Anish Karnik 21110098

GitHub Repository Link: https://github.com/ayushmodi12/CN_Assignment-2/tree/main

Part I: Implement the routing functionality in mininet.

a. **Implementation** of the custom topology using mininet. (Every host should be able to send packets to every other host. You are to submit the code, as well as provide screenshots as proof of its working)

→

Code GitHub Link:

https://github.com/ayushmodi12/CN_Assignment-2/blob/main/part1.py

Screenshots:

pingall:

```
mininet> pingall
*** Ping: testing ping reachability
h1 -> h2 h3 h4 h5 h6 ra rb rc
h2 -> h1 h3 h4 h5 h6 ra rb rc
h3 -> h1 h2 h4 h5 h6 ra rb rc
h4 -> h1 h2 h3 h5 h6 ra rb rc
h5 -> h1 h2 h3 h4 h6 ra rb rc
h6 -> h1 h2 h3 h4 h5 ra rb rc
ra -> h1 h2 h3 h4 h5 h6 rb rc
rb -> h1 h2 h3 h4 h5 h6 ra rc
rc -> h1 h2 h3 h4 h5 h6 ra rb
*** Results: 0% dropped (72/72 received)
```

Pinging the hosts and routers individually:

- h1 ping h2: h1 and h2 present in the same subnet

```
mininet> h1 ping h2
PING 10.0.0.3 (10.0.0.3) 56(84) bytes of data.
64 bytes from 10.0.0.3: icmp_seq=1 ttl=64 time=1.53 ms
64 bytes from 10.0.0.3: icmp_seq=2 ttl=64 time=0.927 ms
64 bytes from 10.0.0.3: icmp_seq=3 ttl=64 time=0.256 ms
64 bytes from 10.0.0.3: icmp_seq=4 ttl=64 time=0.082 ms
64 bytes from 10.0.0.3: icmp_seq=5 ttl=64 time=0.043 ms
64 bytes from 10.0.0.3: icmp_seq=6 ttl=64 time=0.044 ms
64 bytes from 10.0.0.3: icmp_seq=7 ttl=64 time=0.046 ms
64 bytes from 10.0.0.3: icmp_seq=8 ttl=64 time=0.045 ms
64 bytes from 10.0.0.3: icmp_seq=9 ttl=64 time=0.048 ms
64 bytes from 10.0.0.3: icmp_seq=10 ttl=64 time=0.047 ms
64 bytes from 10.0.0.3: icmp_seq=11 ttl=64 time=0.042 ms
^C
--- 10.0.0.3 ping statistics ---
11 packets transmitted, 11 received, 0% packet loss, time 10203ms
rtt min/avg/max/mdev = 0.042/0.282/1.525/0.466 ms
```

- h1 ping h3: h1 and h3 present in different subnets

```
mininet> h1 ping h3
PING 10.1.0.2 (10.1.0.2) 56(84) bytes of data.
64 bytes from 10.1.0.2: icmp_seq=1 ttl=62 time=2.01 ms
64 bytes from 10.1.0.2: icmp_seq=2 ttl=62 time=0.911 ms
64 bytes from 10.1.0.2: icmp_seq=3 ttl=62 time=0.192 ms
64 bytes from 10.1.0.2: icmp_seq=4 ttl=62 time=0.061 ms
64 bytes from 10.1.0.2: icmp_seq=5 ttl=62 time=0.061 ms
64 bytes from 10.1.0.2: icmp_seq=6 ttl=62 time=0.062 ms
64 bytes from 10.1.0.2: icmp_seq=7 ttl=62 time=0.059 ms
64 bytes from 10.1.0.2: icmp_seq=8 ttl=62 time=0.061 ms
64 bytes from 10.1.0.2: icmp_seq=9 ttl=62 time=0.080 ms
64 bytes from 10.1.0.2: icmp_seq=10 ttl=62 time=0.063 ms
64 bytes from 10.1.0.2: icmp_seq=11 ttl=62 time=0.064 ms
^C
--- 10.1.0.2 ping statistics ---
11 packets transmitted, 11 received, 0% packet loss, time 10187ms
rtt min/avg/max/mdev = 0.059/0.329/2.012/0.584 ms
```

- h1 ping h6: h1 and h6 present in different subnets

```
mininet> h1 ping h6
PING 10.2.0.3 (10.2.0.3) 56(84) bytes of data.
64 bytes from 10.2.0.3: icmp_seq=1 ttl=62 time=1.70 ms
64 bytes from 10.2.0.3: icmp_seq=2 ttl=62 time=1.05 ms
64 bytes from 10.2.0.3: icmp_seq=3 ttl=62 time=0.188 ms
64 bytes from 10.2.0.3: icmp_seq=4 ttl=62 time=0.070 ms
64 bytes from 10.2.0.3: icmp_seq=5 ttl=62 time=0.062 ms
64 bytes from 10.2.0.3: icmp_seq=6 ttl=62 time=0.064 ms
64 bytes from 10.2.0.3: icmp_seq=7 ttl=62 time=0.059 ms
64 bytes from 10.2.0.3: icmp_seq=8 ttl=62 time=0.060 ms
64 bytes from 10.2.0.3: icmp_seq=9 ttl=62 time=0.059 ms
64 bytes from 10.2.0.3: icmp_seq=10 ttl=62 time=0.063 ms
64 bytes from 10.2.0.3: icmp_seq=11 ttl=62 time=0.083 ms
^C
--- 10.2.0.3 ping statistics ---
11 packets transmitted, 11 received, 0% packet loss, time 10206ms
rtt min/avg/max/mdev = 0.059/0.314/1.697/0.519 ms
```

- h4 ping h5: h4 and h5 present in different subnets

```
mininet> h4 ping h5
PING 10.2.0.2 (10.2.0.2) 56(84) bytes of data.
64 bytes from 10.2.0.2: icmp_seq=1 ttl=62 time=1.99 ms
64 bytes from 10.2.0.2: icmp_seq=2 ttl=62 time=0.303 ms
64 bytes from 10.2.0.2: icmp_seq=3 ttl=62 time=0.059 ms
64 bytes from 10.2.0.2: icmp_seq=4 ttl=62 time=0.059 ms
64 bytes from 10.2.0.2: icmp_seq=5 ttl=62 time=0.059 ms
64 bytes from 10.2.0.2: icmp_seq=6 ttl=62 time=0.059 ms
64 bytes from 10.2.0.2: icmp_seq=7 ttl=62 time=0.058 ms
64 bytes from 10.2.0.2: icmp_seq=8 ttl=62 time=0.059 ms
64 bytes from 10.2.0.2: icmp_seq=9 ttl=62 time=0.057 ms
64 bytes from 10.2.0.2: icmp_seq=10 ttl=62 time=0.056 ms
64 bytes from 10.2.0.2: icmp_seq=11 ttl=62 time=0.062 ms
^C
--- 10.2.0.2 ping statistics ---
11 packets transmitted, 11 received, 0% packet loss, time 10244ms
rtt min/avg/max/mdev = 0.056/0.256/1.988/0.552 ms
```

- h5 ping h2: h5 and h2 present in different subnets

```
mininet> h5 ping h2
PING 10.0.0.3 (10.0.0.3) 56(84) bytes of data.
64 bytes from 10.0.0.3: icmp_seq=1 ttl=62 time=2.28 ms
64 bytes from 10.0.0.3: icmp_seq=2 ttl=62 time=2.28 ms
64 bytes from 10.0.0.3: icmp_seq=3 ttl=62 time=0.247 ms
64 bytes from 10.0.0.3: icmp_seq=4 ttl=62 time=0.053 ms
64 bytes from 10.0.0.3: icmp_seq=5 ttl=62 time=0.092 ms
64 bytes from 10.0.0.3: icmp_seq=6 ttl=62 time=0.034 ms
64 bytes from 10.0.0.3: icmp_seq=7 ttl=62 time=0.059 ms
64 bytes from 10.0.0.3: icmp_seq=8 ttl=62 time=0.061 ms
64 bytes from 10.0.0.3: icmp_seq=9 ttl=62 time=0.061 ms
64 bytes from 10.0.0.3: icmp_seq=10 ttl=62 time=0.106 ms
64 bytes from 10.0.0.3: icmp_seq=11 ttl=62 time=0.062 ms
^C
--- 10.0.0.3 ping statistics ---
11 packets transmitted, 11 received, 0% packet loss, time 10201ms
rtt min/avg/max/mdev = 0.034/0.484/2.279/0.847 ms
```

- ra ping rb:

```
mininet> ra ping rb
PING 10.1.0.1 (10.1.0.1) 56(84) bytes of data.
64 bytes from 10.1.0.1: icmp_seq=1 ttl=64 time=0.033 ms
64 bytes from 10.1.0.1: icmp_seq=2 ttl=64 time=0.033 ms
64 bytes from 10.1.0.1: icmp_seq=3 ttl=64 time=0.036 ms
64 bytes from 10.1.0.1: icmp_seq=4 ttl=64 time=0.041 ms
64 bytes from 10.1.0.1: icmp_seq=5 ttl=64 time=0.039 ms
64 bytes from 10.1.0.1: icmp_seq=6 ttl=64 time=0.037 ms
64 bytes from 10.1.0.1: icmp_seq=7 ttl=64 time=0.048 ms
64 bytes from 10.1.0.1: icmp_seq=8 ttl=64 time=0.034 ms
64 bytes from 10.1.0.1: icmp_seq=9 ttl=64 time=0.036 ms
64 bytes from 10.1.0.1: icmp_seq=10 ttl=64 time=0.041 ms
64 bytes from 10.1.0.1: icmp_seq=11 ttl=64 time=0.039 ms
^C
--- 10.1.0.1 ping statistics ---
11 packets transmitted, 11 received, 0% packet loss, time 10227ms
rtt min/avg/max/mdev = 0.033/0.037/0.048/0.004 ms
```

- rc ping ra:

```
mininet> rc ping ra
PING 10.0.0.1 (10.0.0.1) 56(84) bytes of data.
64 bytes from 10.0.0.1: icmp_seq=1 ttl=64 time=0.031 ms
64 bytes from 10.0.0.1: icmp_seq=2 ttl=64 time=0.038 ms
64 bytes from 10.0.0.1: icmp_seq=3 ttl=64 time=0.034 ms
64 bytes from 10.0.0.1: icmp_seq=4 ttl=64 time=0.037 ms
64 bytes from 10.0.0.1: icmp_seq=5 ttl=64 time=0.035 ms
64 bytes from 10.0.0.1: icmp_seq=6 ttl=64 time=0.036 ms
64 bytes from 10.0.0.1: icmp_seq=7 ttl=64 time=0.036 ms
64 bytes from 10.0.0.1: icmp_seq=8 ttl=64 time=0.035 ms
64 bytes from 10.0.0.1: icmp_seq=9 ttl=64 time=0.036 ms
64 bytes from 10.0.0.1: icmp_seq=10 ttl=64 time=0.035 ms
64 bytes from 10.0.0.1: icmp_seq=11 ttl=64 time=0.038 ms
^C
--- 10.0.0.1 ping statistics ---
11 packets transmitted, 11 received, 0% packet loss, time 10274ms
rtt min/avg/max/mdev = 0.031/0.035/0.038/0.002 ms
```

- h2 ping rc:

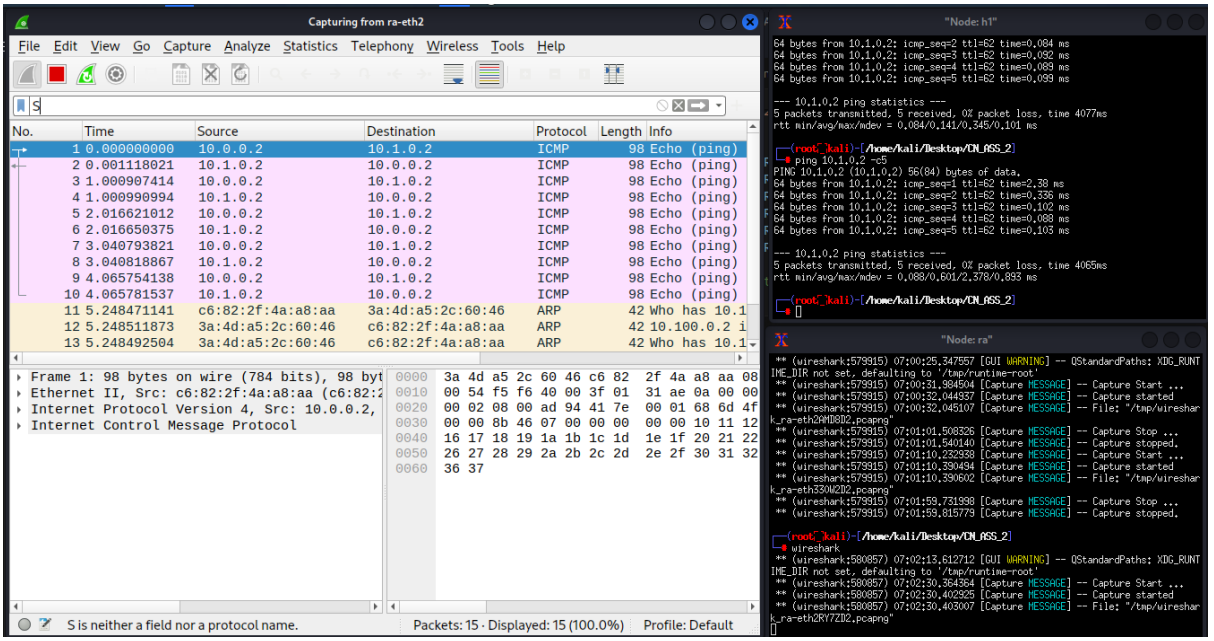
```
mininet> h2 ping rc
PING 10.2.0.1 (10.2.0.1) 56(84) bytes of data.
64 bytes from 10.2.0.1: icmp_seq=1 ttl=63 time=1.19 ms
64 bytes from 10.2.0.1: icmp_seq=2 ttl=63 time=0.369 ms
64 bytes from 10.2.0.1: icmp_seq=3 ttl=63 time=0.053 ms
64 bytes from 10.2.0.1: icmp_seq=4 ttl=63 time=0.052 ms
64 bytes from 10.2.0.1: icmp_seq=5 ttl=63 time=0.052 ms
64 bytes from 10.2.0.1: icmp_seq=6 ttl=63 time=0.053 ms
64 bytes from 10.2.0.1: icmp_seq=7 ttl=63 time=0.057 ms
64 bytes from 10.2.0.1: icmp_seq=8 ttl=63 time=0.051 ms
64 bytes from 10.2.0.1: icmp_seq=9 ttl=63 time=0.054 ms
64 bytes from 10.2.0.1: icmp_seq=10 ttl=63 time=0.052 ms
64 bytes from 10.2.0.1: icmp_seq=11 ttl=63 time=0.051 ms
^C
--- 10.2.0.1 ping statistics ---
11 packets transmitted, 11 received, 0% packet loss, time 10207ms
rtt min/avg/max/mdev = 0.051/0.185/1.194/0.331 ms
```

- ra ping h6:

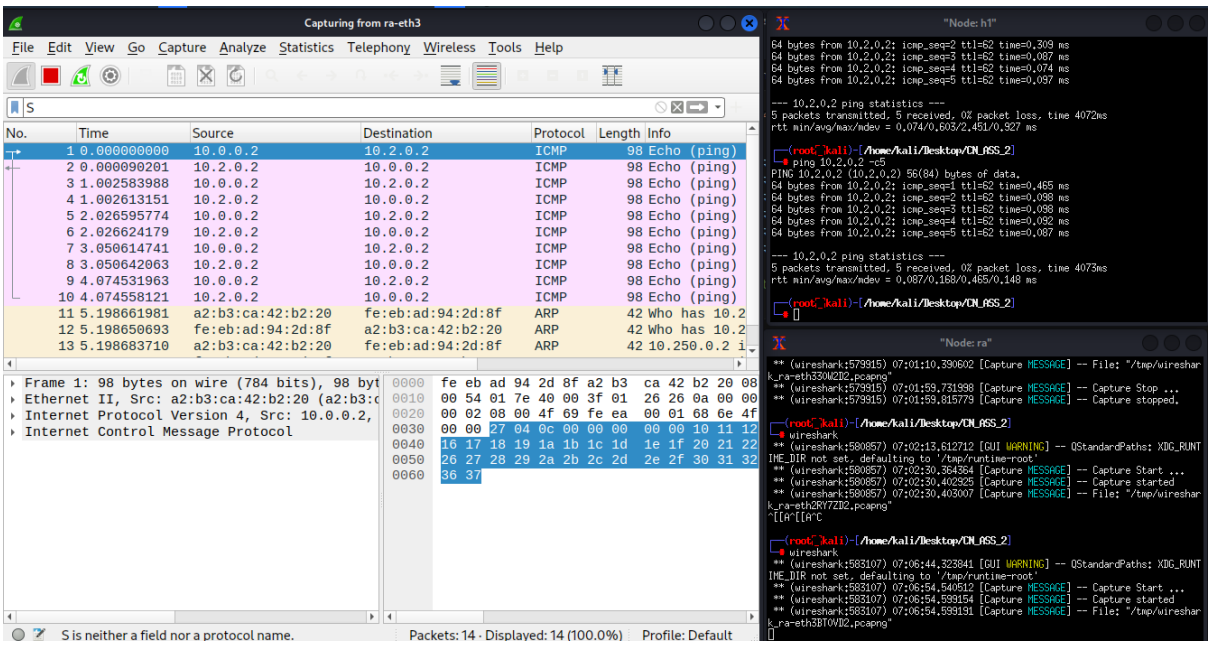
```
mininet> ra ping h6
PING 10.2.0.3 (10.2.0.3) 56(84) bytes of data.
64 bytes from 10.2.0.3: icmp_seq=1 ttl=63 time=1.01 ms
64 bytes from 10.2.0.3: icmp_seq=2 ttl=63 time=0.826 ms
64 bytes from 10.2.0.3: icmp_seq=3 ttl=63 time=0.230 ms
64 bytes from 10.2.0.3: icmp_seq=4 ttl=63 time=0.042 ms
64 bytes from 10.2.0.3: icmp_seq=5 ttl=63 time=0.055 ms
64 bytes from 10.2.0.3: icmp_seq=6 ttl=63 time=0.070 ms
64 bytes from 10.2.0.3: icmp_seq=7 ttl=63 time=0.052 ms
64 bytes from 10.2.0.3: icmp_seq=8 ttl=63 time=0.052 ms
64 bytes from 10.2.0.3: icmp_seq=9 ttl=63 time=0.053 ms
64 bytes from 10.2.0.3: icmp_seq=10 ttl=63 time=0.055 ms
64 bytes from 10.2.0.3: icmp_seq=11 ttl=63 time=0.055 ms
^C
--- 10.2.0.3 ping statistics ---
11 packets transmitted, 11 received, 0% packet loss, time 10704ms
rtt min/avg/max/mdev = 0.042/0.226/1.006/0.330 ms
```

b. **Observations**: Capture and show the wireshark/tcpdump (packets) for the route setup on any one of the routers.

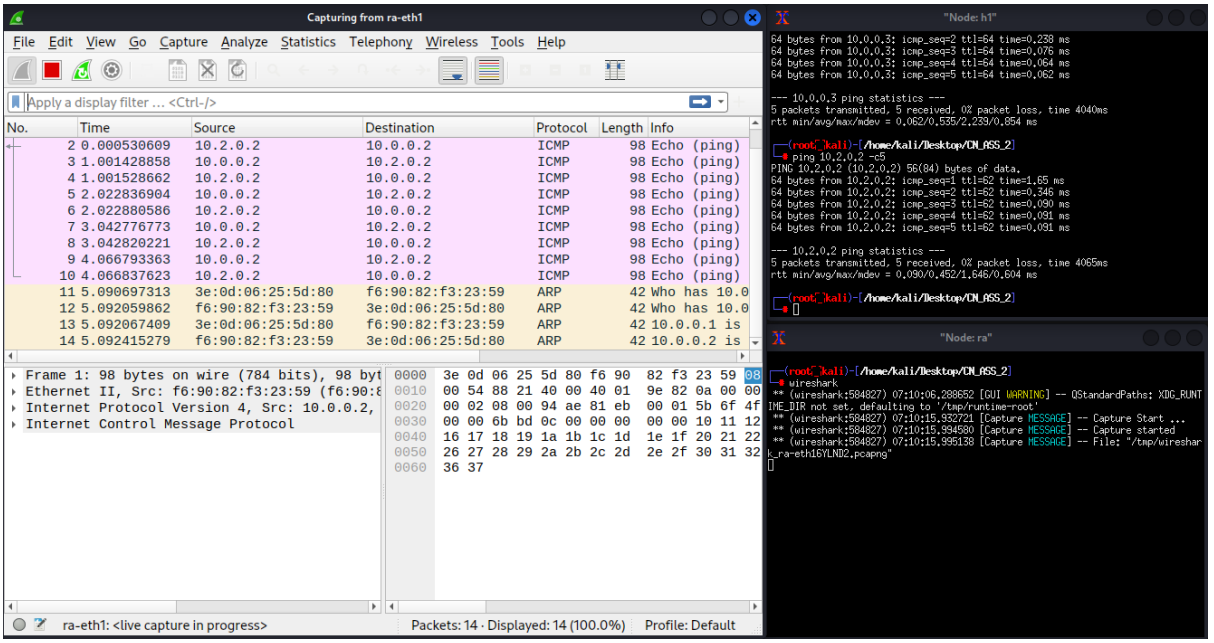
→



h1->ra->rb->h3 (h1->h3) Packet capture for Router A(ra-eth2)



h1->ra->rc->h5 (h1->h5) Packet capture for Router A(ra-eth3)



h1->ra->rc->h5 (h1->h5) Packet capture for Router A(ra-eth1)

c. **Vary the default routing and measure the latency difference. (i.e. the default route for a packet from h1 to h6 would be h1 -> r_a -> r_c -> h6. Try to vary the path so that it takes the path h1-> r_a -> r_b -> r_c -> h6. Provide the screenshots as proof for the latency difference, both ping as well as iperf).**

Code GitHub Link:

https://github.com/ayushmodi12/CN_Assignment-2/blob/main/part1_qc.py

For finding latency using ping we used cmd to find details of 5 packets that we sent. The latency can be taken as a measure of average Round trip time. For iperf, we sent packets for 5s and found the bandwidth and transfer for these paths.

- For Path h1 -> r_a -> r_c -> h6

```
Ping Latency Measurement (ra-> rc):
PING 10.2.0.3 (10.2.0.3) 56(84) bytes of data.
64 bytes from 10.2.0.3: icmp_seq=1 ttl=62 time=17.9 ms
64 bytes from 10.2.0.3: icmp_seq=2 ttl=62 time=0.424 ms
64 bytes from 10.2.0.3: icmp_seq=3 ttl=62 time=0.087 ms
64 bytes from 10.2.0.3: icmp_seq=4 ttl=62 time=0.081 ms
64 bytes from 10.2.0.3: icmp_seq=5 ttl=62 time=0.074 ms

--- 10.2.0.3 ping statistics ---
5 packets transmitted, 5 received, 0% packet loss, time 4051ms
rtt min/avg/max/mdev = 0.074/3.705/17.862/7.079 ms
Traceroute Path (ra-> rc):
traceroute to 10.2.0.3 (10.2.0.3), 30 hops max, 60 byte packets
 1  10.0.0.1 (10.0.0.1)  3.462 ms  3.646 ms  4.068 ms
 2  10.250.0.1 (10.250.0.1)  4.340 ms  4.582 ms  5.141 ms
 3  * * 10.2.0.3 (10.2.0.3)  8.985 ms
*** Measuring latency using iperf (ra-> rc):
Iperf Latency Measurement (ra-> rc):
------------------------------------------------------------
Client connecting to 10.2.0.3, TCP port 5001
TCP window size: 85.3 KByte (default)
------------------------------------------------------------
[  1] local 10.0.0.2 port 34322 connected with 10.2.0.3 port 5001 (icwnd/mss/irtt=14/1448/5173)
[ ID] Interval       Transfer     Bandwidth
[  1] 0.0000-5.0013 sec  19.1 GBytes  32.9 Gbits/sec
```

Average RTT for this path is : 3.705ms. Also implemented traceroute to trace the path of packets. Bandwidth is also greater for the direct path than the indirect one.

- For path h1-> r_a -> r_b -> r_c -> h6:

```
Ping Latency Measurement (ra-> rc):
PING 10.2.0.3 (10.2.0.3) 56(84) bytes of data.
64 bytes from 10.2.0.3: icmp_seq=1 ttl=61 time=36.1 ms
64 bytes from 10.2.0.3: icmp_seq=2 ttl=61 time=0.185 ms
64 bytes from 10.2.0.3: icmp_seq=3 ttl=61 time=0.093 ms
64 bytes from 10.2.0.3: icmp_seq=4 ttl=61 time=0.093 ms
64 bytes from 10.2.0.3: icmp_seq=5 ttl=61 time=0.078 ms

--- 10.2.0.3 ping statistics ---
5 packets transmitted, 5 received, 0% packet loss, time 4077ms
rtt min/avg/max/mdev = 0.078/7.307/36.087/14.389 ms
Traceroute Path (ra-> rc):
traceroute to 10.2.0.3 (10.2.0.3), 30 hops max, 60 byte packets
 1  10.0.0.1 (10.0.0.1)  3.240 ms  3.755 ms  3.778 ms
 2  10.100.0.2 (10.100.0.2)  3.783 ms  3.785 ms  3.788 ms
 3  10.200.0.2 (10.200.0.2)  3.791 ms  3.793 ms  3.793 ms
 4  10.2.0.3 (10.2.0.3)  4.859 ms  4.784 ms  4.793 ms
*** Measuring latency using iperf (ra-> rc):
Iperf Latency Measurement (ra-> rc):
------------------------------------------------------------
Client connecting to 10.2.0.3, TCP port 5001
TCP window size: 85.3 KByte (default)
------------------------------------------------------------
[  1] local 10.0.0.2 port 45914 connected with 10.2.0.3 port 5001 (icwnd/mss/irtt=14/1448/3675)
[ ID] Interval       Transfer     Bandwidth
[  1] 0.0000-5.0050 sec  13.6 GBytes  23.3 Gbits/sec
```

Average RTT for this path is : 7.307ms. The bandwidth is lower for these path than the direct one.

d. Dump the routing tables for all the routers for questions a & c.

→

- The Routing Tables for question a:

Path: ra->rc

```
*** Routing Tables on Routers:
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 ra-eth1
10.1.0.0        10.100.0.2      255.255.255.0   UG    0      0        0 ra-eth2
10.2.0.0        10.250.0.1      255.255.255.0   UG    0      0        0 ra-eth3
10.100.0.0      0.0.0.0         255.255.255.0   U     0      0        0 ra-eth2
10.250.0.0      0.0.0.0         255.255.255.0   U     0      0        0 ra-eth3
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 rb-eth2
10.1.0.0        0.0.0.0         255.255.255.0   U     0      0        0 rb-eth1
10.2.0.0        10.200.0.2      255.255.255.0   UG    0      0        0 rb-eth3
10.100.0.0      0.0.0.0         255.255.255.0   U     0      0        0 rb-eth2
10.200.0.0      0.0.0.0         255.255.255.0   U     0      0        0 rb-eth3
Kernel IP routing table
Destination     Gateway         Genmask         Flags Metric Ref    Use Iface
10.0.0.0        10.250.0.2      255.255.255.0   UG    0      0        0 rc-eth3
10.1.0.0        10.200.0.1      255.255.255.0   UG    0      0        0 rc-eth2
10.2.0.0        0.0.0.0         255.255.255.0   U     0      0        0 rc-eth1
10.200.0.0      0.0.0.0         255.255.255.0   U     0      0        0 rc-eth2
10.250.0.0      0.0.0.0         255.255.255.0   U     0      0        0 rc-eth3
```

- The Routing Tables for question c:

Path: ra->rb->rc

```
*** Routing Tables on Routers:
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 ra-eth1
10.1.0.0        10.100.0.2      255.255.255.0   UG    0      0        0 ra-eth2
10.2.0.0        10.100.0.2      255.255.255.0   UG    0      0        0 ra-eth2
10.100.0.0      0.0.0.0         255.255.255.0   U     0      0        0 ra-eth2
10.250.0.0      0.0.0.0         255.255.255.0   U     0      0        0 ra-eth3
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 rb-eth2
10.1.0.0        0.0.0.0         255.255.255.0   U     0      0        0 rb-eth1
10.2.0.0        10.200.0.2      255.255.255.0   UG    0      0        0 rb-eth3
10.100.0.0      0.0.0.0         255.255.255.0   U     0      0        0 rb-eth2
10.200.0.0      0.0.0.0         255.255.255.0   U     0      0        0 rb-eth3
Kernel IP routing table
Destination     Gateway         Genmask         Flags Metric Ref    Use Iface
10.0.0.0        10.200.0.1      255.255.255.0   UG    0      0        0 rc-eth2
10.1.0.0        10.200.0.1      255.255.255.0   UG    0      0        0 rc-eth2
10.2.0.0        0.0.0.0         255.255.255.0   U     0      0        0 rc-eth1
10.200.0.0      0.0.0.0         255.255.255.0   U     0      0        0 rc-eth2
10.250.0.0      0.0.0.0         255.255.255.0   U     0      0        0 rc-eth3
```

Part II: Throughput for different congestion control schemes.

a. Implementation of the mininet topology and TCP client-server program (for packet generation).

The TCP client-server program should run with the required configuration for question (b) when --config=b and with required configuration for question (c) when --config=c

Other configurable parameters like congestion control scheme, link loss should also be taken as command line arguments to the program.

→

[IMP]

NOTE:
To give the config: b or c, congestion control scheme and the link loss, in our code, we take input from the user from the terminal after running the python script.

In the following way:

```
└─$ sudo python part2.py
```

And then give the input of the parameters in the following way:

```
└─$ sudo python part2.py
[sudo] password for kali:
Enter configuration (b/c): b
Enter congestion control scheme: reno
Enter link loss percentage: 0
```

Code GitHub Link:

https://github.com/ayushmodi12/CN_Assignment-2/blob/main/part2.py

b. Run the client on H1 and the server on H4. Report and reason the throughput

over time for each congestion control scheme.

→

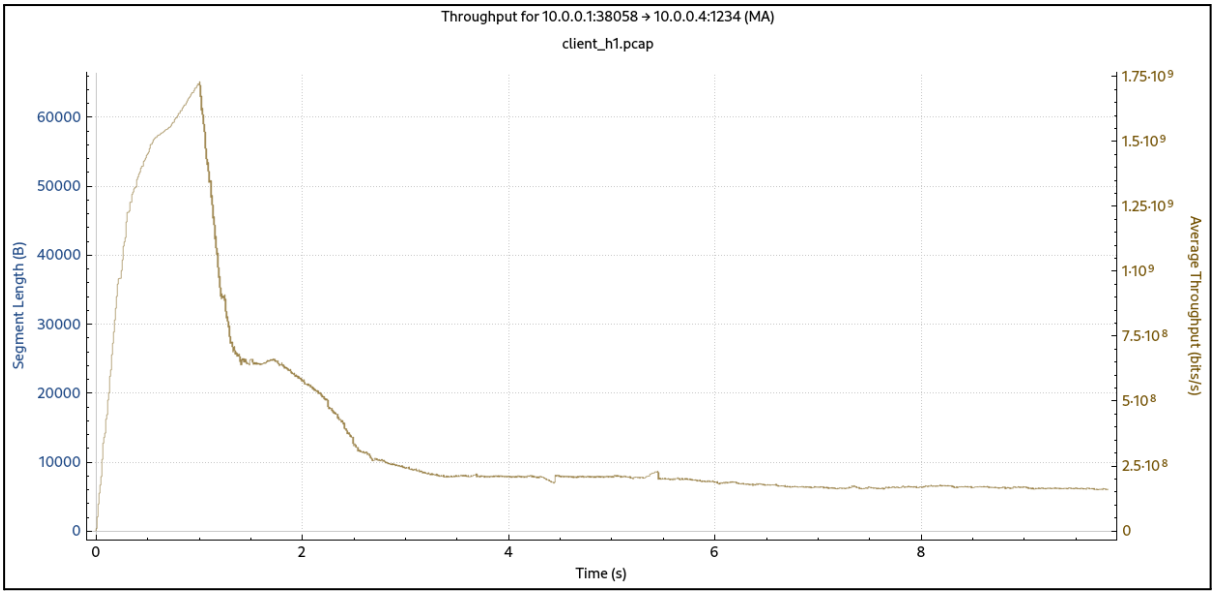
Throughput graphs for each congestion control scheme:

H1 is ip: 10.0.0.1
H2 is ip: 10.0.0.2
H3 is ip: 10.0.0.3
H4 is ip: 10.0.0.4

- Congestion Control Scheme: **TCP Vegas**

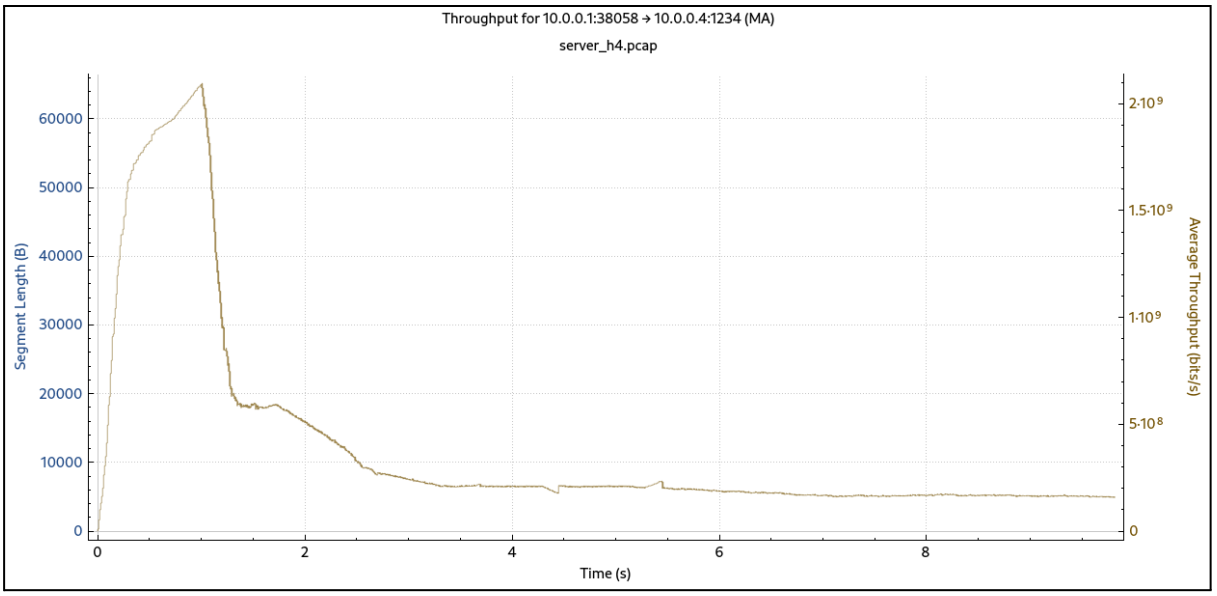
```
Enter configuration (b/c): b
Enter congestion control scheme: vegas
Enter link loss percentage: 0
*** Creating network
*** Adding controller
*** Adding hosts:
h1 h2 h3 h4
*** Adding switches:
s1 s2
*** Adding links:
(s1, h1) (s1, h2) (s1, s2) (s2, h3) (s2, h4)
*** Configuring hosts
h1 h2 h3 h4
*** Starting controller
c0
*** Starting 2 switches
s1 s2 ...
*** Starting CLI:
```

For client H1:



TCP Vegas is a rate/delay based congestion control scheme. Vegas congestion control graph is pretty smooth as it performs gradual and not sudden changes in throughput and thus the curves have very less sudden drops or spikes in throughput.

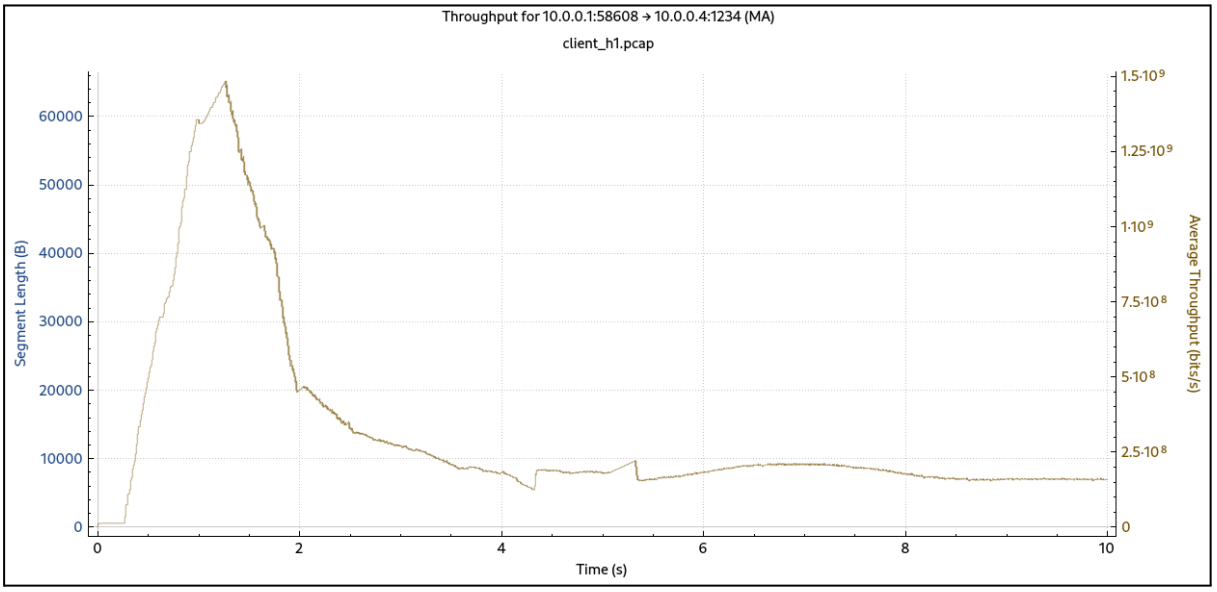
For server H4:



- Congestion Control Scheme: **TCP Reno**

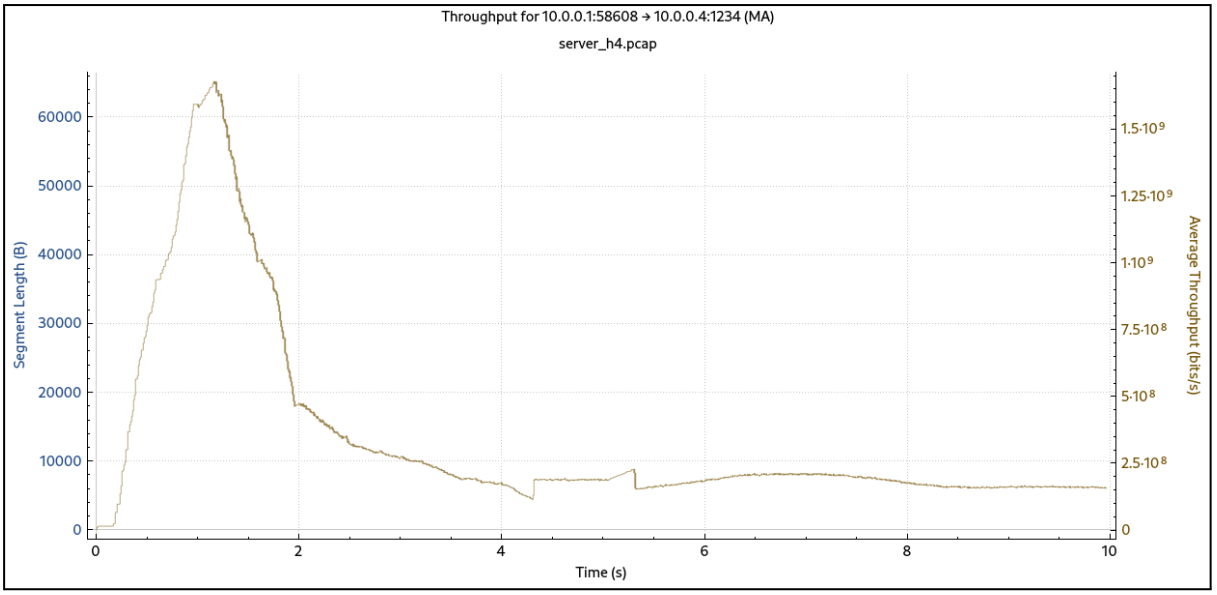
```
Enter configuration (b/c): b
Enter congestion control scheme: reno
Enter link loss percentage: 0
*** Creating network
*** Adding controller
*** Adding hosts:
h1 h2 h3 h4
*** Adding switches:
s1 s2
*** Adding links:
(s1, h1) (s1, h2) (s1, s2) (s2, h3) (s2, h4)
*** Configuring hosts
h1 h2 h3 h4
*** Starting controller
c0
*** Starting 2 switches
s1 s2 ...
*** Starting CLI:
mininet>
```

For client H1:



As we can see here Reno utilizes linear function to increase size. We also know that Reno changes the segment size to half if there is a loss of packet by using a fast recovery algorithm that quickly recovers from packet loss. Also, in comparison to Vegas, Reno has fast recovery and retransmission for handling packet loss. Thus we observe a slightly aggressive increase in the congestion window after timeout. At the same time, the graph is also not as smooth as Vegas, it is slightly jerky.

For server H4:

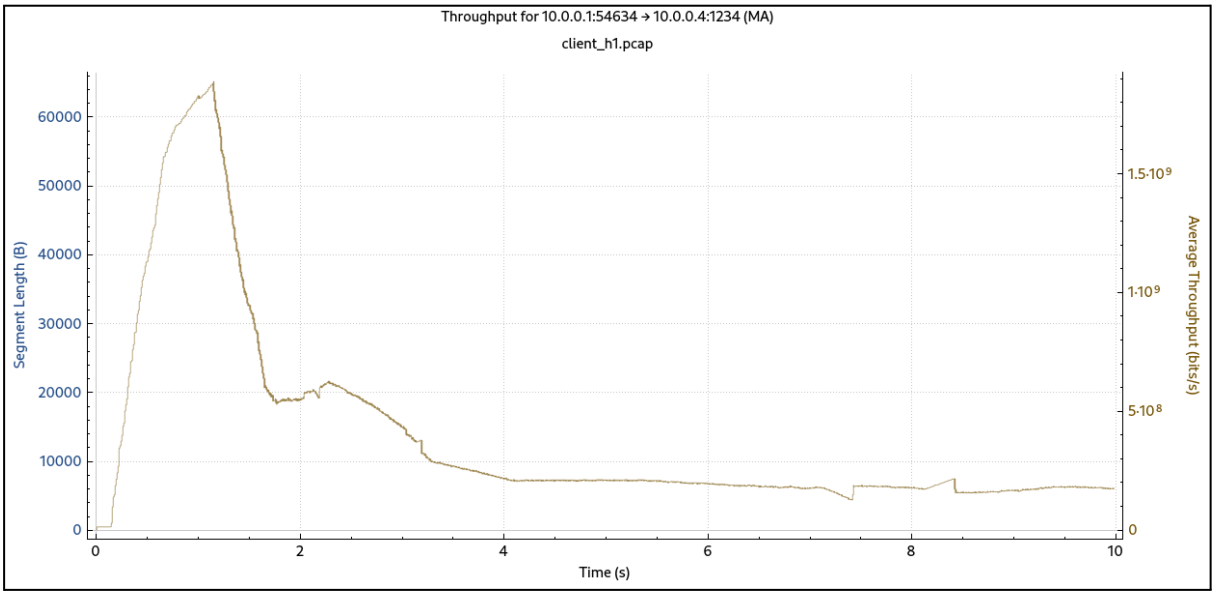


For a single source we would receive the same number of packets as sent by the client. As the congestion control is Reno we are getting linear increase in segment size.

- Congestion Control Scheme: **TCP Cubic**

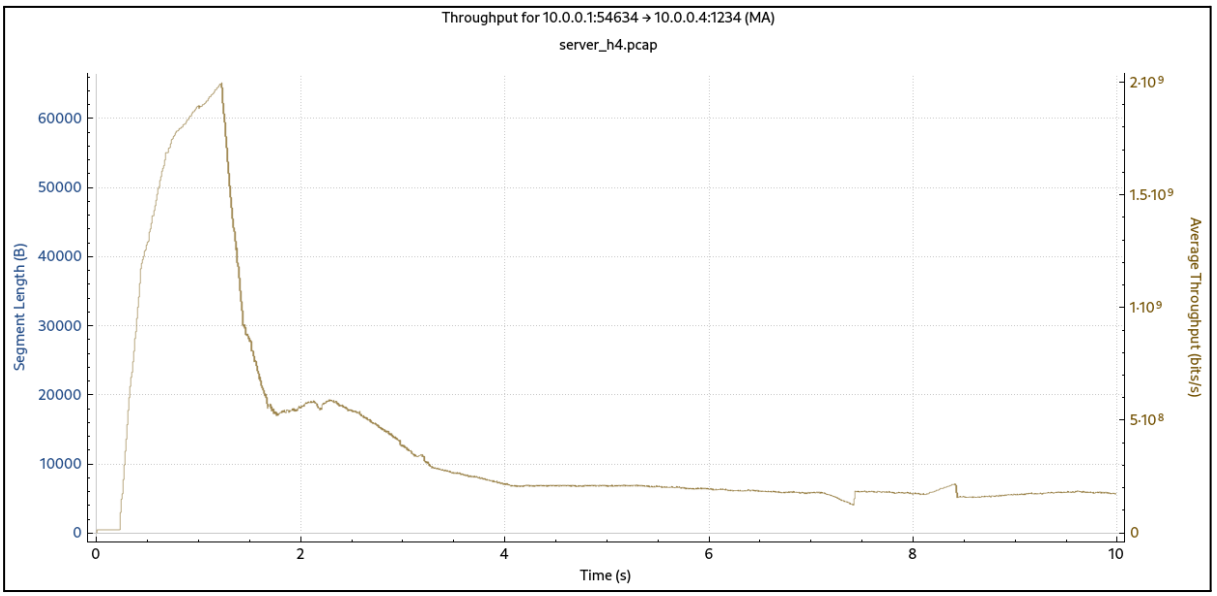
```
Enter configuration (b/c): b
Enter congestion control scheme: cubic
Enter link loss percentage: 0
*** Creating network
*** Adding controller
*** Adding hosts:
h1 h2 h3 h4
*** Adding switches:
s1 s2
*** Adding links:
(s1, h1) (s1, h2) (s1, s2) (s2, h3) (s2, h4)
*** Configuring hosts
h1 h2 h3 h4
*** Starting controller
c0
*** Starting 2 switches
s1 s2 ...
*** Starting CLI:
```

For client H1:



As we can see here Cubic utilizes cubic function to increase size. We can see in the graph that initially, the curve looks like a cubic curve. Also, there is a steeper increase in throughput after bandwidth probing. We also know that Cubic changes the segment size by a smaller factor if there is a loss of packet compared to Reno as we can see above.

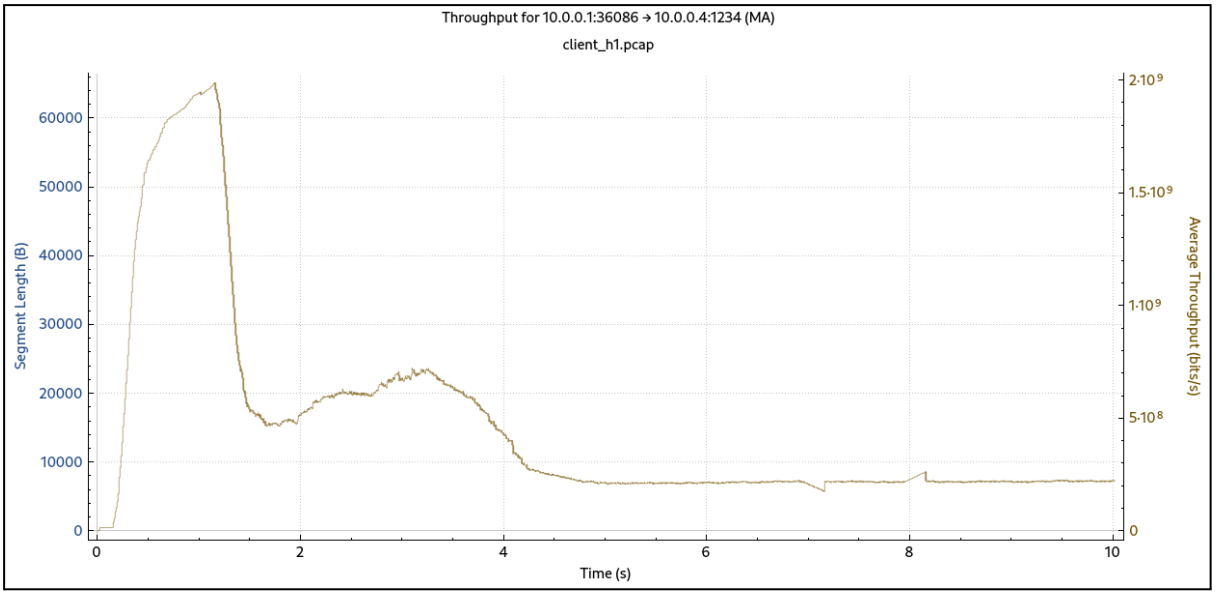
For server H4:



- Congestion Control Scheme: **TCP BBR**

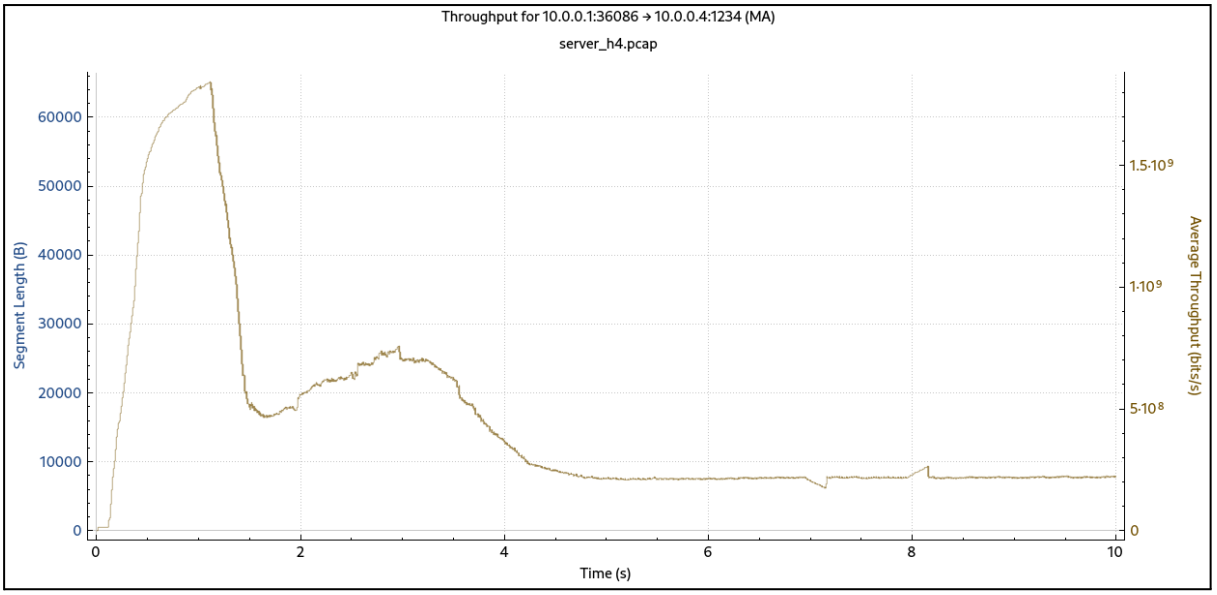
```
Enter configuration (b/c): b
Enter congestion control scheme: bbr
Enter link loss percentage: 0
*** Creating network
*** Adding controller
*** Adding hosts:
h1 h2 h3 h4
*** Adding switches:
s1 s2
*** Adding links:
(s1, h1) (s1, h2) (s1, s2) (s2, h3) (s2, h4)
*** Configuring hosts
h1 h2 h3 h4
*** Starting controller
c0
*** Starting 2 switches
s1 s2 ...
*** Starting CLI:
mininet>
```

For client H1:



BBR has been proved to have higher throughputs. On comparing all the graphs we can see that the overall throughput is greater for this scheme.

For server H4:



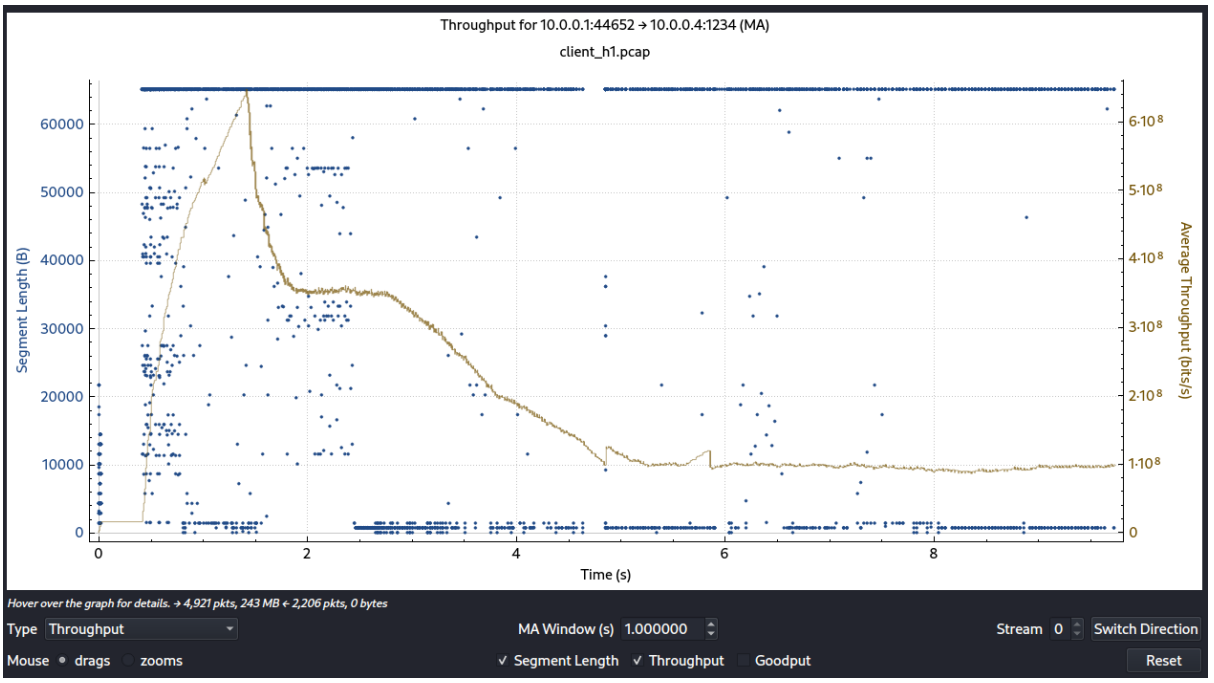
c. Run the client on H1, H2, H3 simultaneously and the server on H4. Report and reason the throughput over time for each host, each congestion control scheme.

In this part (c) , the overall reasoning and comparisons for the throughput graphs remains similar to what we have explained in part (b). Additional point to note is what we can observe in this multi-client setup is the fairness of the tcp congestion control protocols. We could observe from the throughput graphs of the various congestion control schemes that due to there being more than one client, although each client starts with a different throughput initially, at the end they mostly reach a level at which they get more or less similar throughputs. This proves the fairness property of tcp congestion control mechanisms.

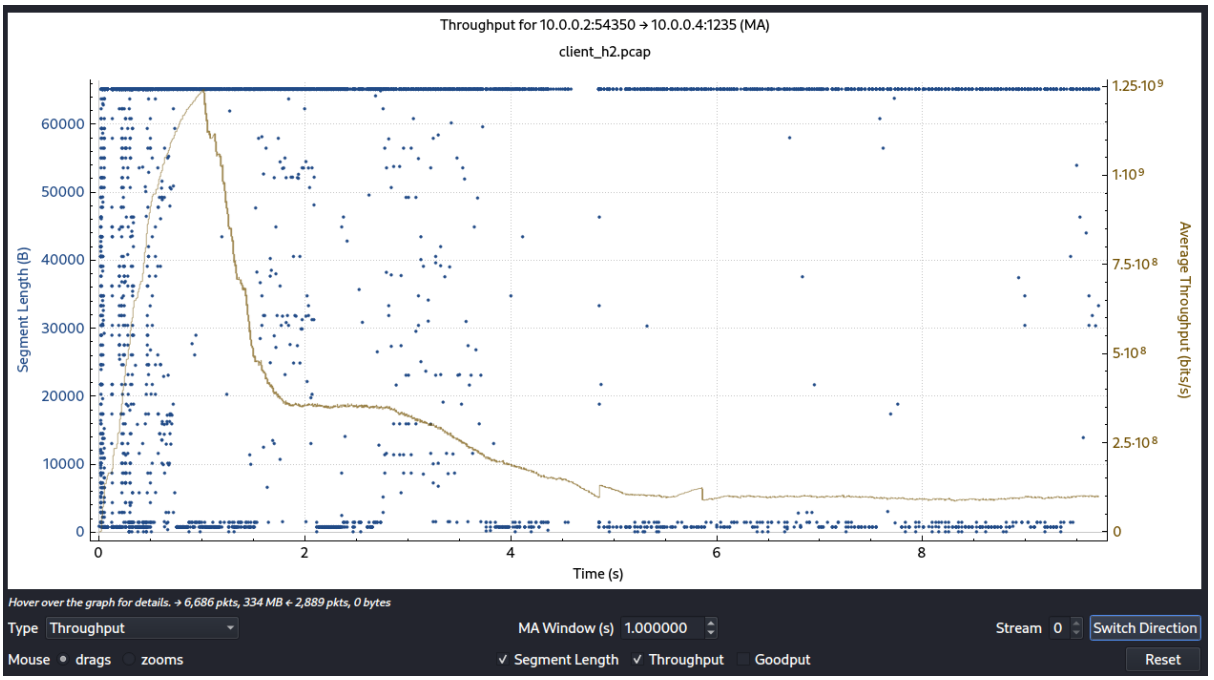
- Congestion Control Scheme: **TCP Vegas**

```
Enter configuration (b/c): c
Enter congestion control scheme: vegas
Enter link loss percentage: 0
*** Creating network
*** Adding controller
*** Adding hosts:
h1 h2 h3 h4
*** Adding switches:
s1 s2
*** Adding links:
(s1, h1) (s1, h2) (s1, s2) (s2, h3) (s2, h4)
*** Configuring hosts
h1 h2 h3 h4
*** Starting controller
c0
*** Starting 2 switches
s1 s2 ...
*** Starting CLI:
mininet>
```

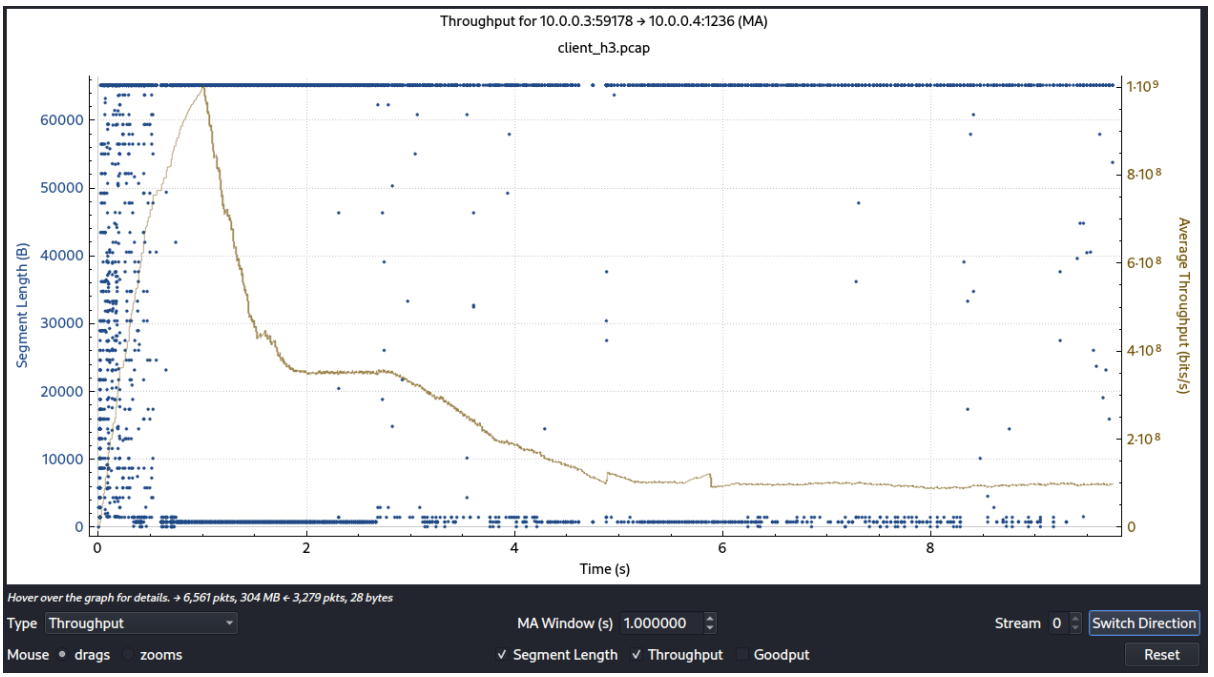
For client H1:



For client H2:

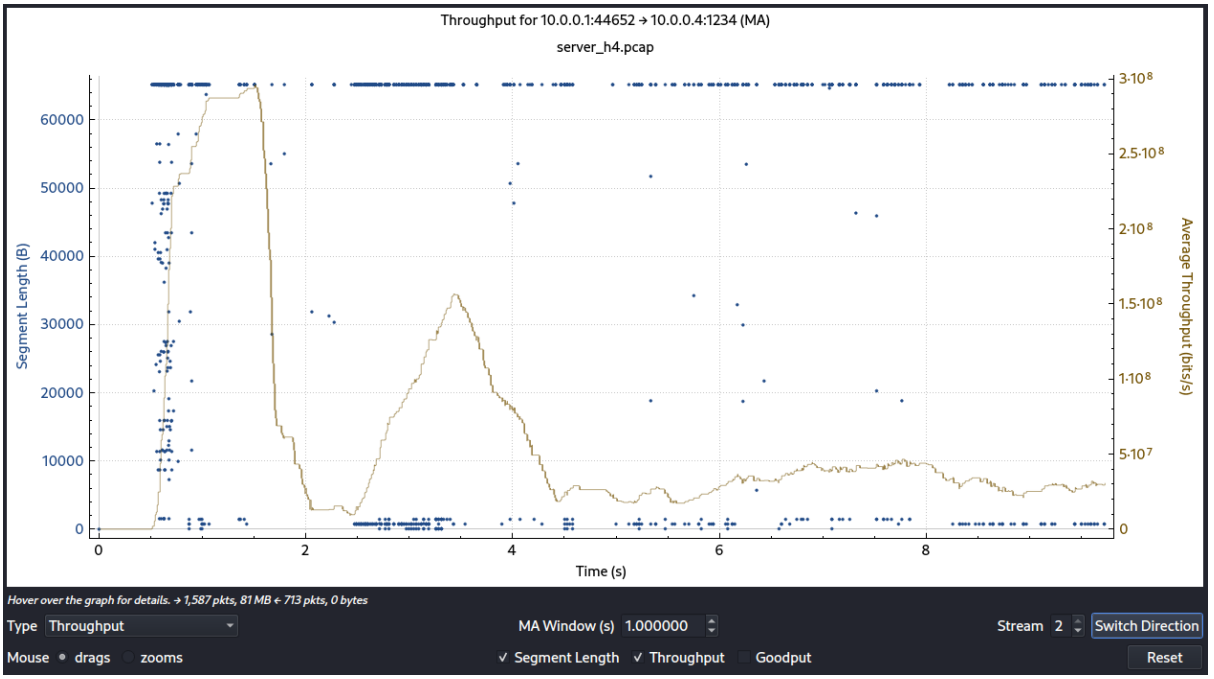


For client H3:

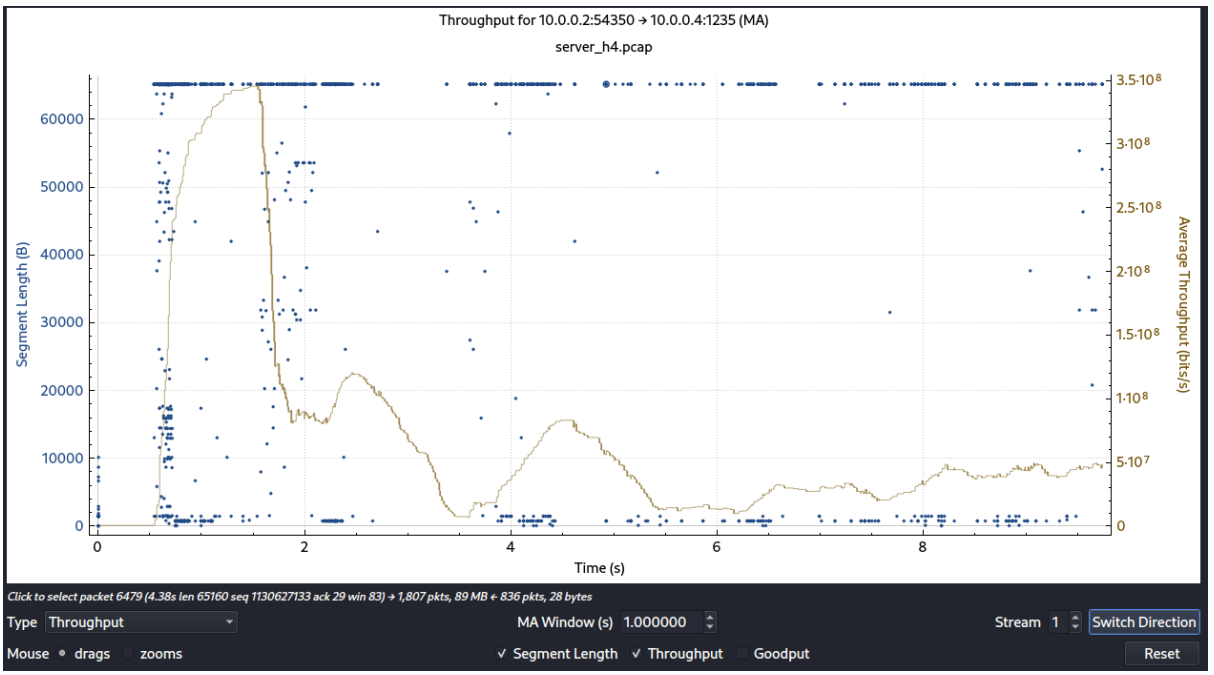


For server H4:

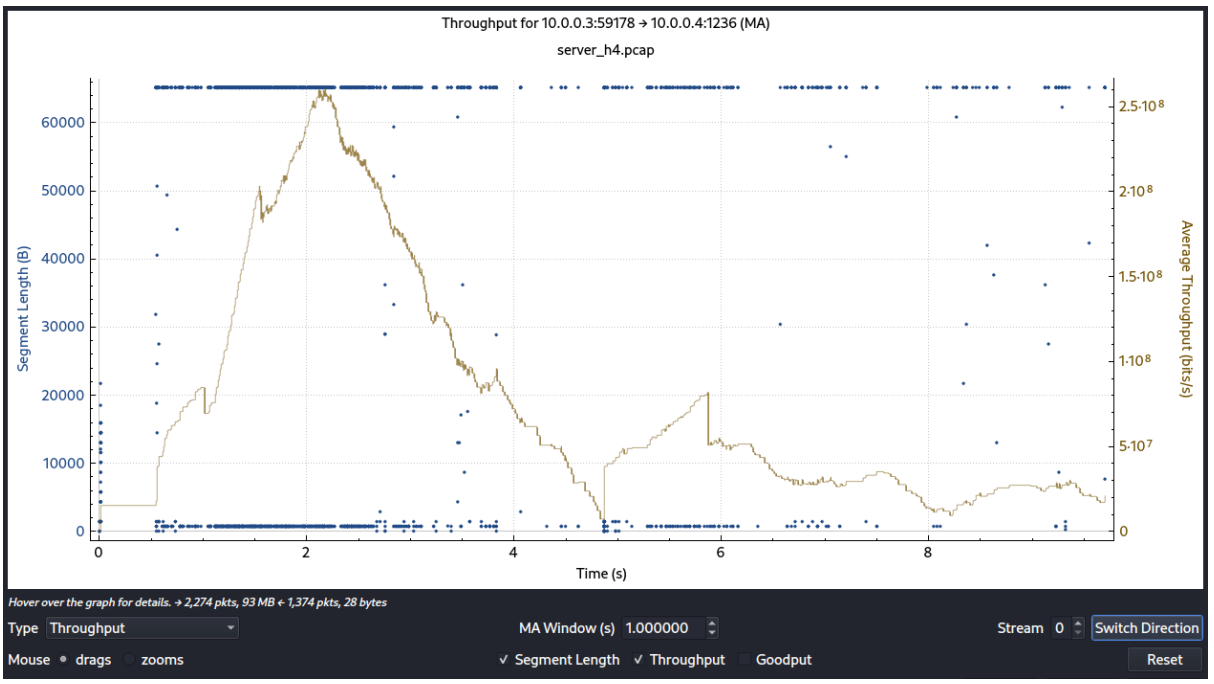
From H1 to H4:



From H2 to H4:



From H3 to H4:

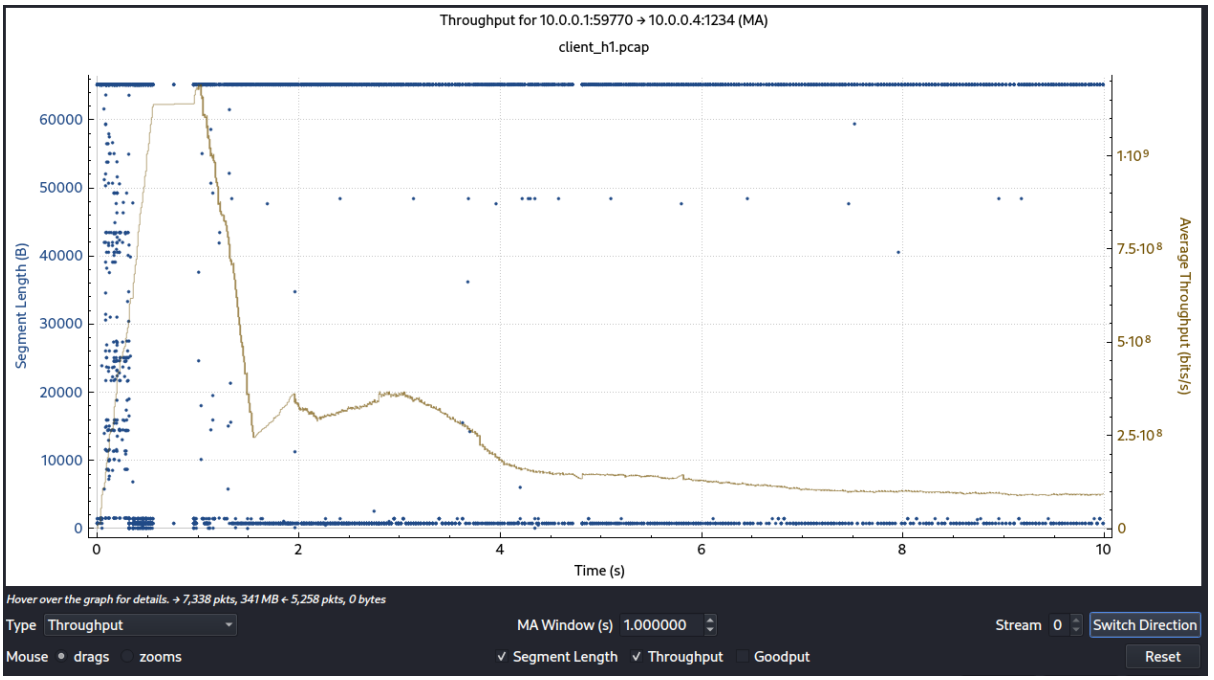


The curve of throughput received by Server H4 from client H3 is different from Client H1 and Client H2, because H1 and H2 send packets through one common switch S1, while H3 is the sole client which is connected to H4 through switch S2. This gives justification for the difference in their graphs.

- Congestion Control Scheme: **TCP Reno**

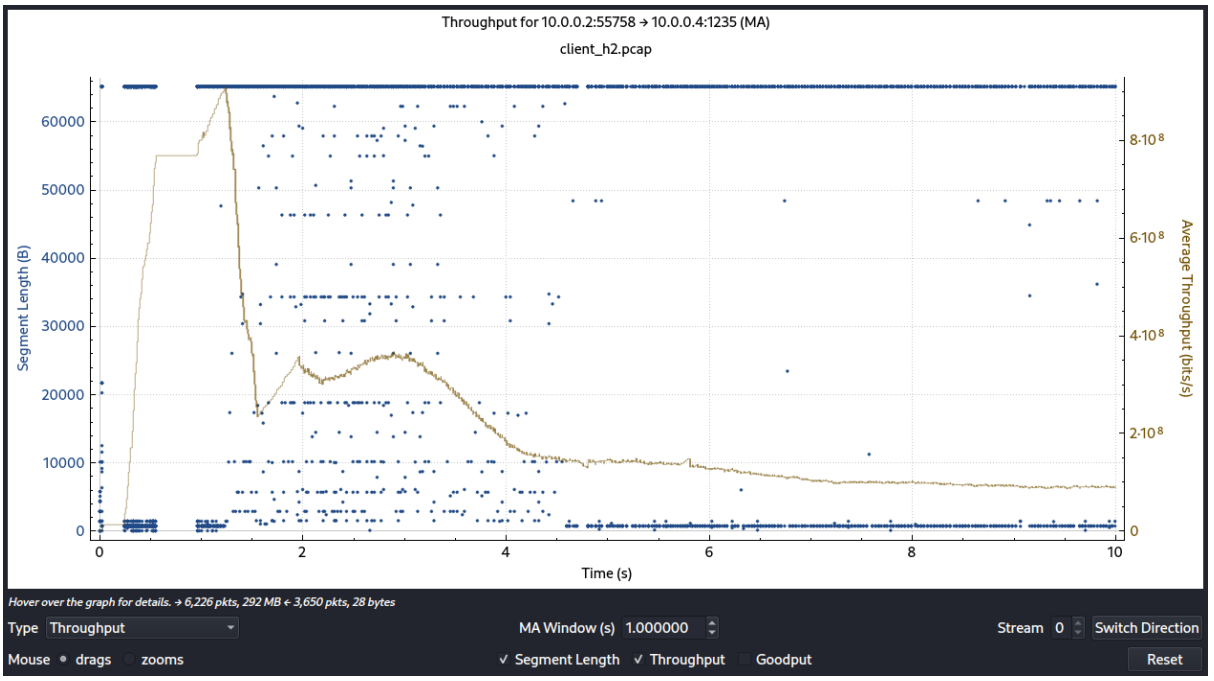
```
Enter configuration (b/c): c
Enter congestion control scheme: reno
Enter link loss percentage: 0
*** Creating network
*** Adding controller
*** Adding hosts:
h1 h2 h3 h4
*** Adding switches:
s1 s2
*** Adding links:
(s1, h1) (s1, h2) (s1, s2) (s2, h3) (s2, h4)
*** Configuring hosts
h1 h2 h3 h4
*** Starting controller
c0
*** Starting 2 switches
s1 s2 ...
*** Starting CLI:
mininet>
```

For client H1:

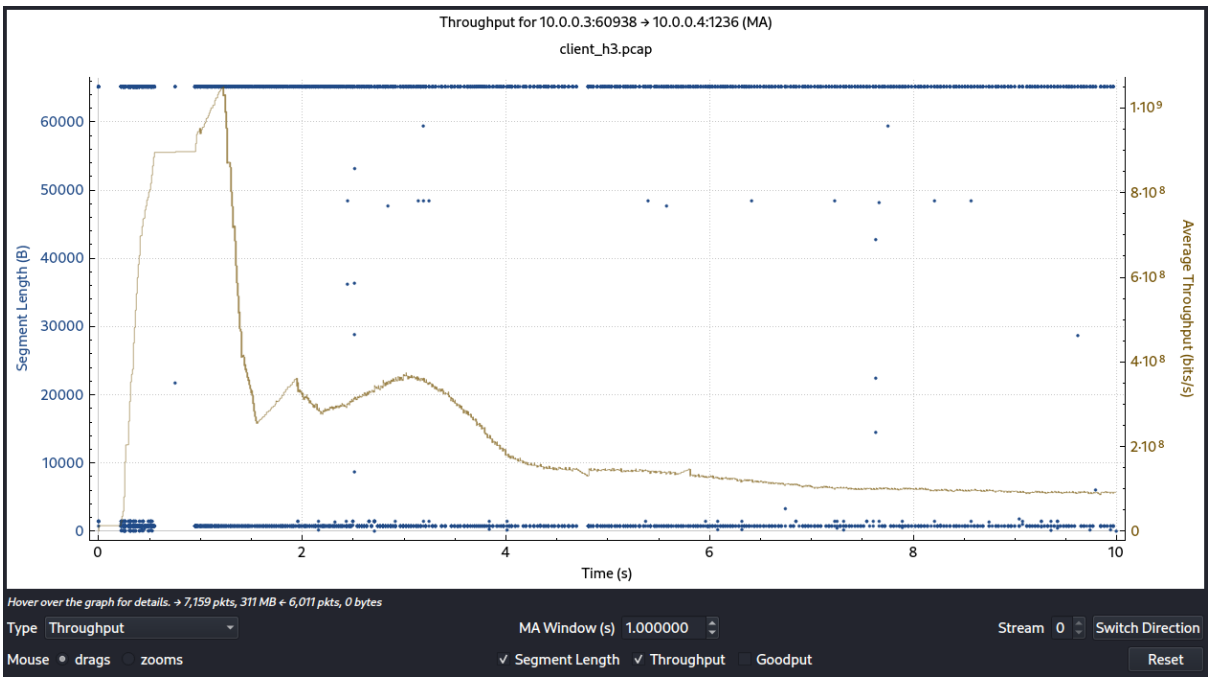


In Reno, the curves have a steep throughput rise initially, and has sudden changes in throughput upon congestion.

For client H2:

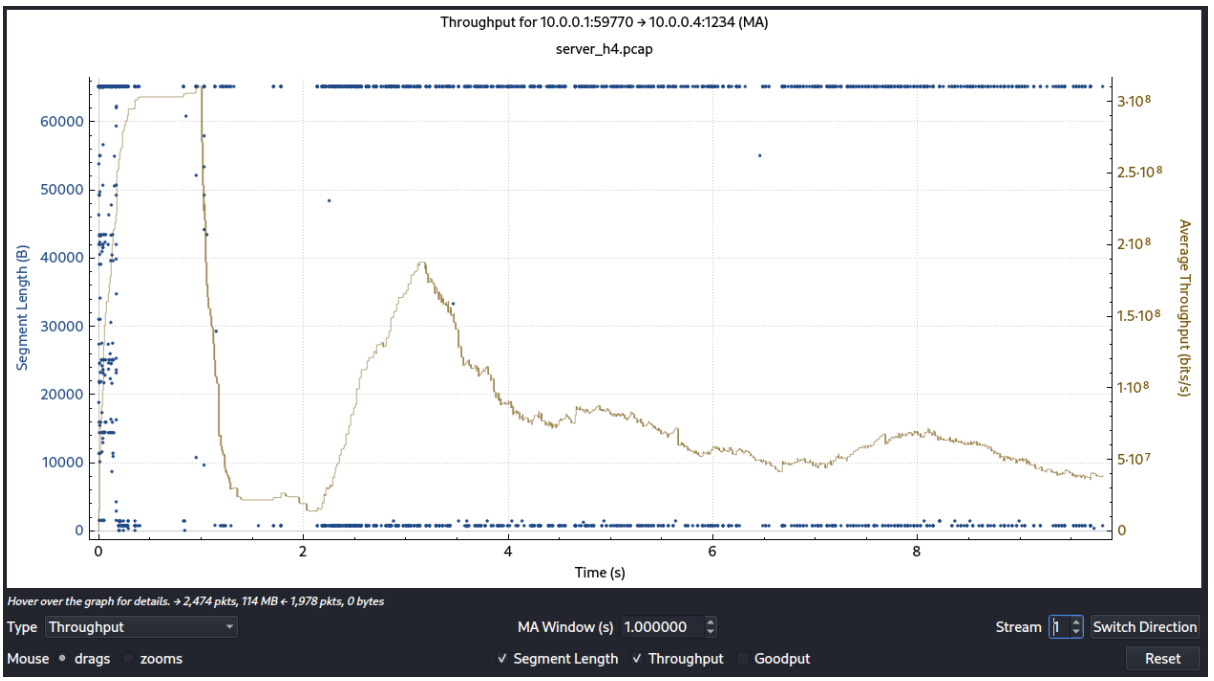


For client H3:

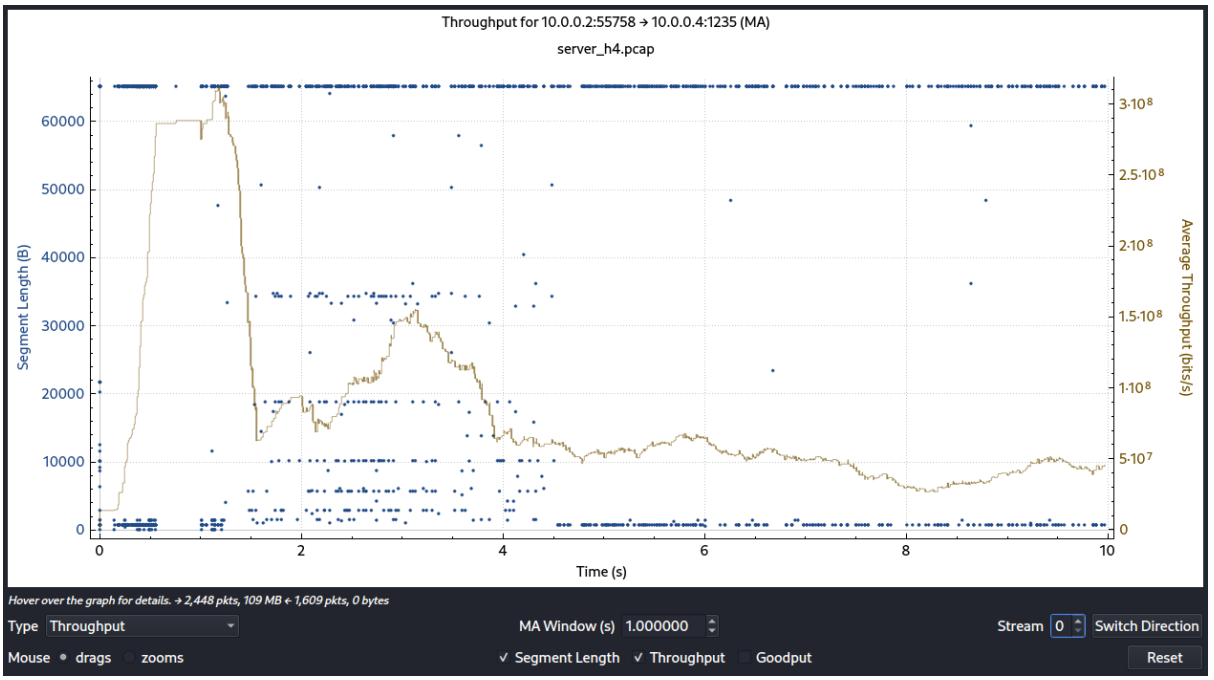


For server H4:

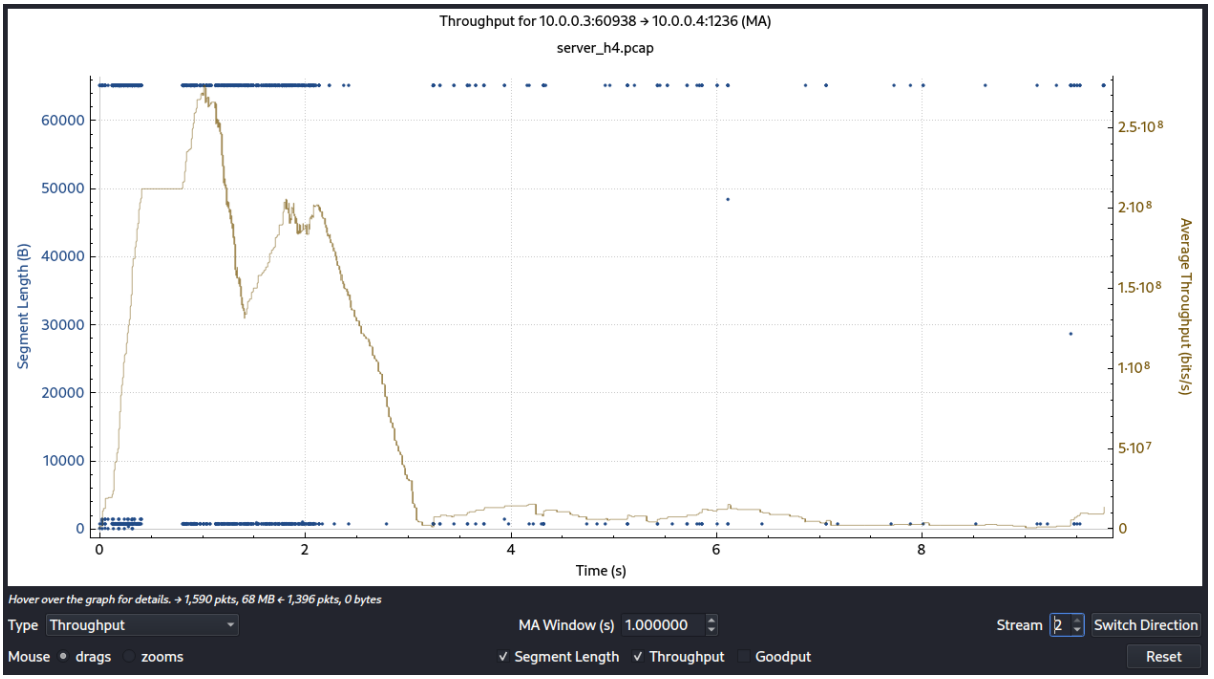
(from H1 to H4):


Throughput for 10.0.0.1:59770 → 10.0.0.4:1234 (MA)
server_h4.pcap
Segment Length (B)
Average Throughput (bits/s)
Time (s)
Hover over the graph for details. → 2,474 pkts, 114 MB ← 1,978 pkts, 0 bytes
Type Throughput
MA Window (s) 1.000000
Stream 1
Switch Direction
Mouse drags zooms
Segment Length
Throughput
Goodput
Reset

From H2 to H4:


Throughput for 10.0.0.2:55758 → 10.0.0.4:1235 (MA)
server_h4.pcap
Segment Length (B)
Average Throughput (bits/s)
Time (s)
Hover over the graph for details. → 2,448 pkts, 109 MB ← 1,609 pkts, 0 bytes
Type Throughput
MA Window (s) 1.000000
Stream 0
Switch Direction
Mouse drags zooms
Segment Length
Throughput
Goodput
Reset

From H3 to H4:

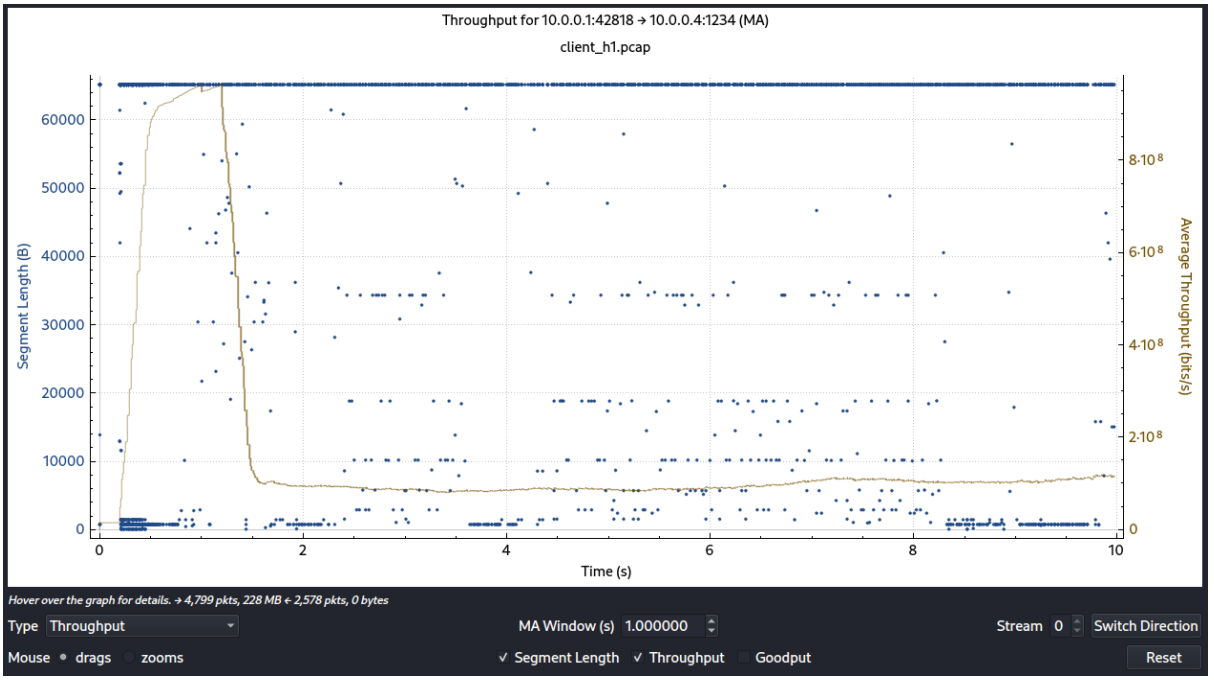


Again it is observed that the curve for throughput of H4 received from H3 is different from that from H2 and H1.

- Congestion Control Scheme: **TCP Cubic**

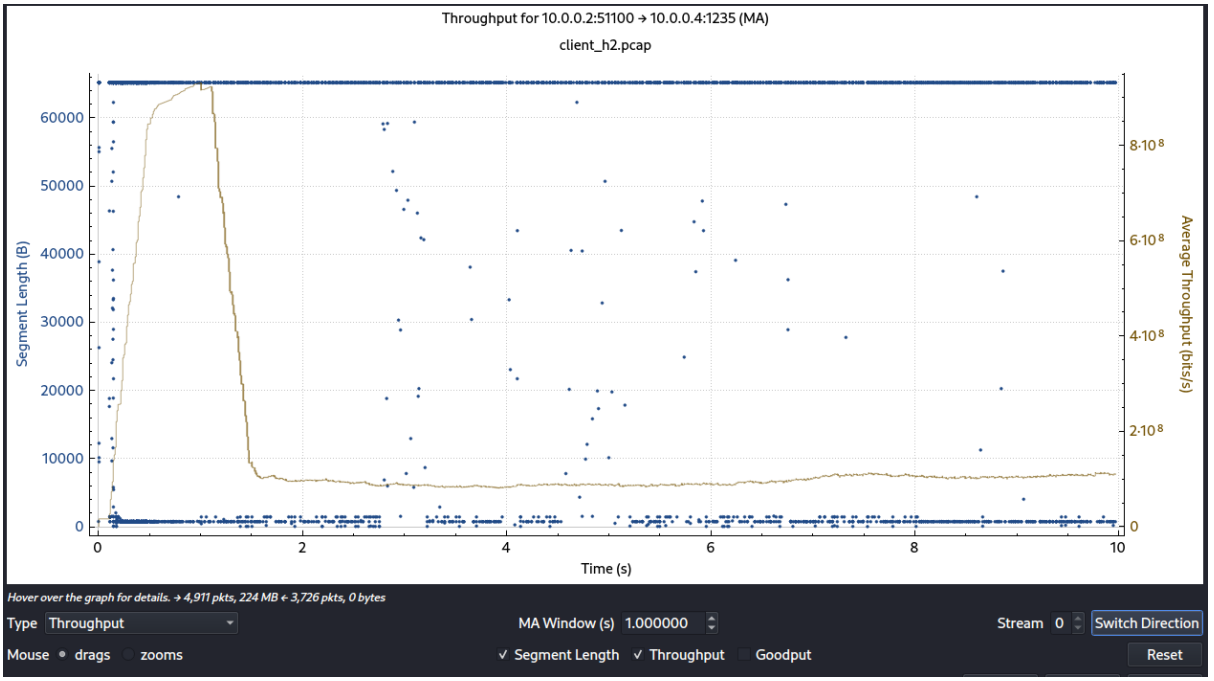
```
Enter configuration (b/c): c
Enter congestion control scheme: cubic
Enter link loss percentage: 0
*** Creating network
*** Adding controller
*** Adding hosts:
h1 h2 h3 h4
*** Adding switches:
s1 s2
*** Adding links:
(s1, h1) (s1, h2) (s1, s2) (s2, h3) (s2, h4)
*** Configuring hosts
h1 h2 h3 h4
*** Starting controller
c0
*** Starting 2 switches
s1 s2 ...
*** Starting CLI:
mininet>
```

For Client H1:

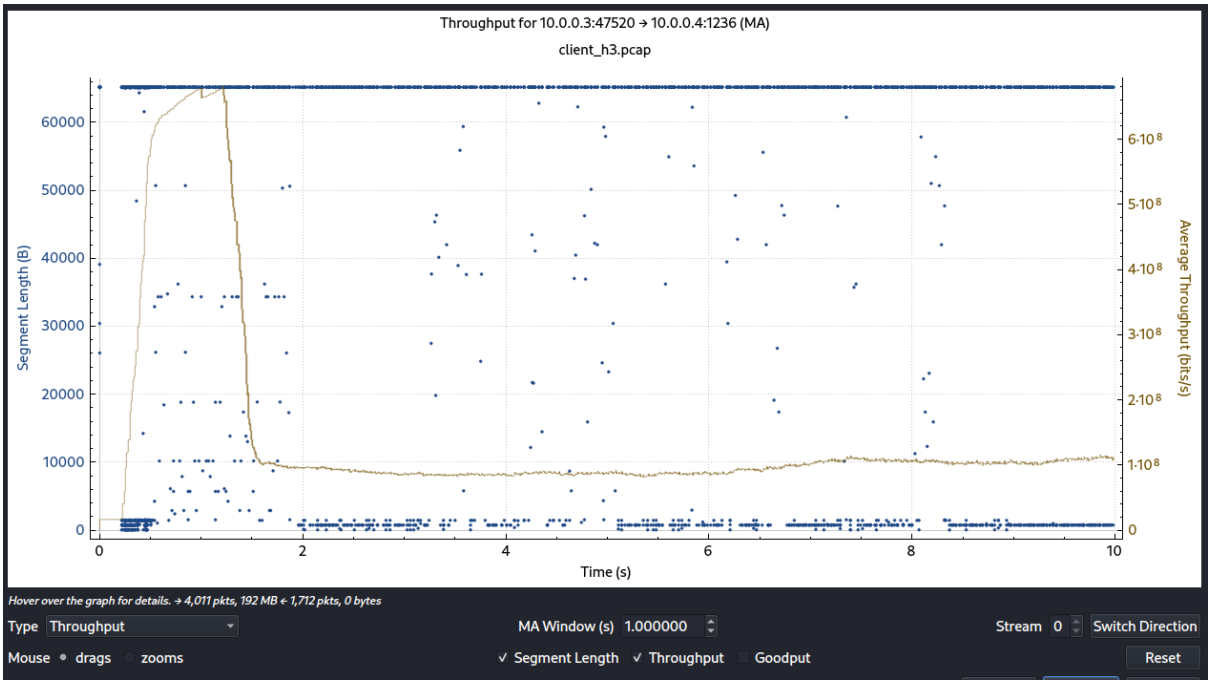


The curve for cubic is significantly different from the other congestion control schemes. It is like a cubic function initially, then it gets capped when it reaches the limit, and then it suddenly decreases after which it remains almost constant for the remainder of the time. Only tcp cubic exhibits such a graph.

For Client H2:

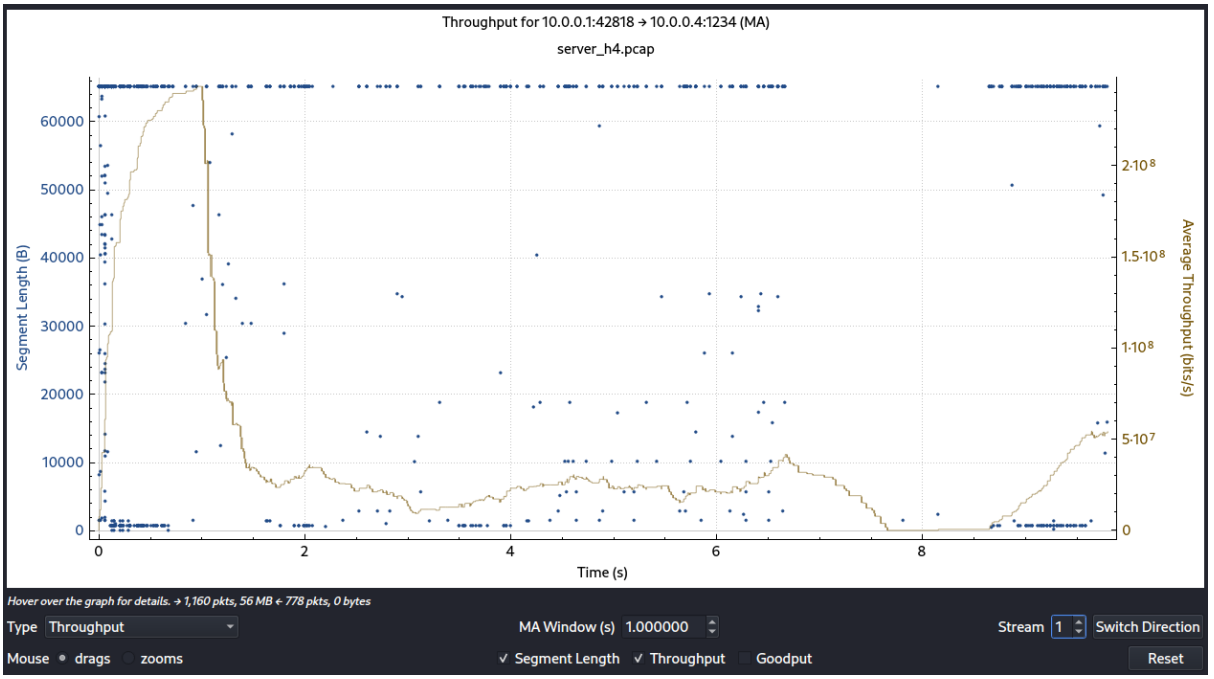


For Client H3:

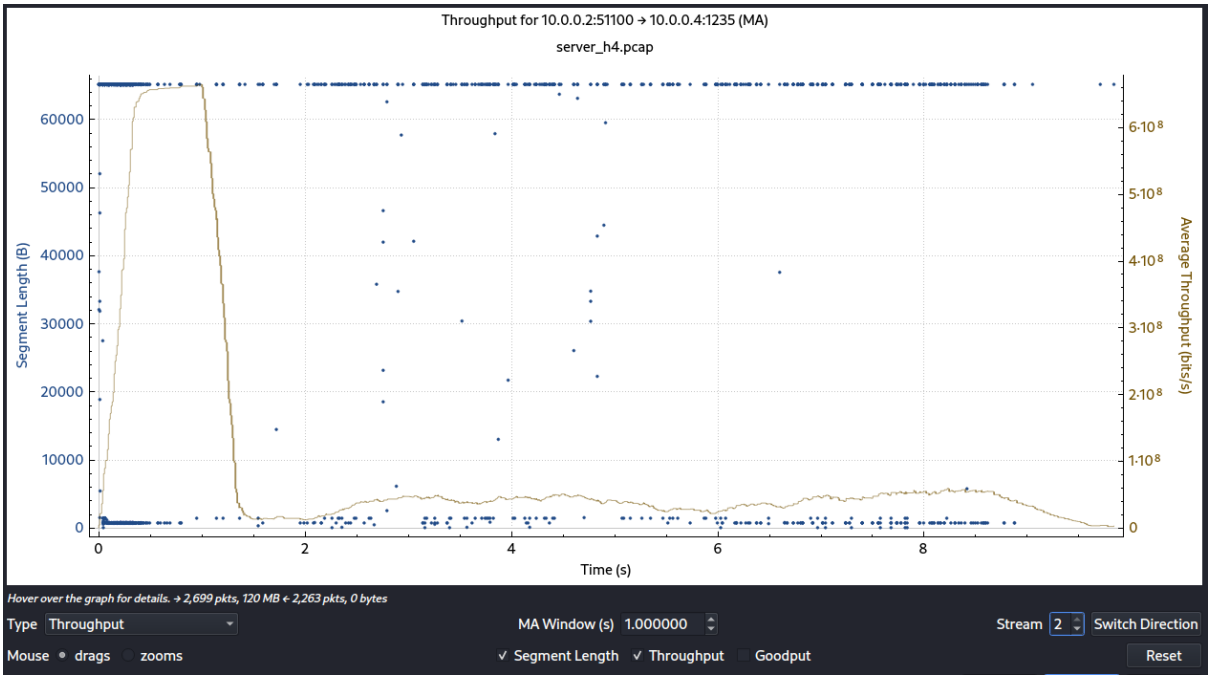


For Server H4:

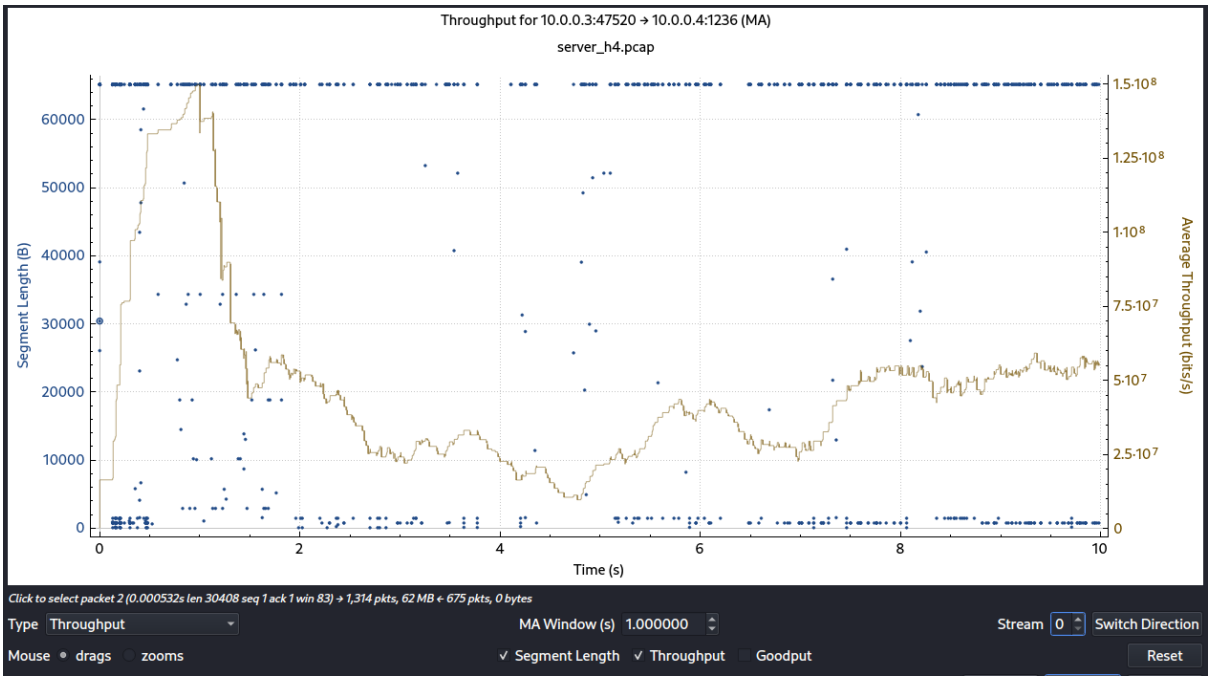
From H1 to H4:



From H2 to H4:



From H3 to H4:

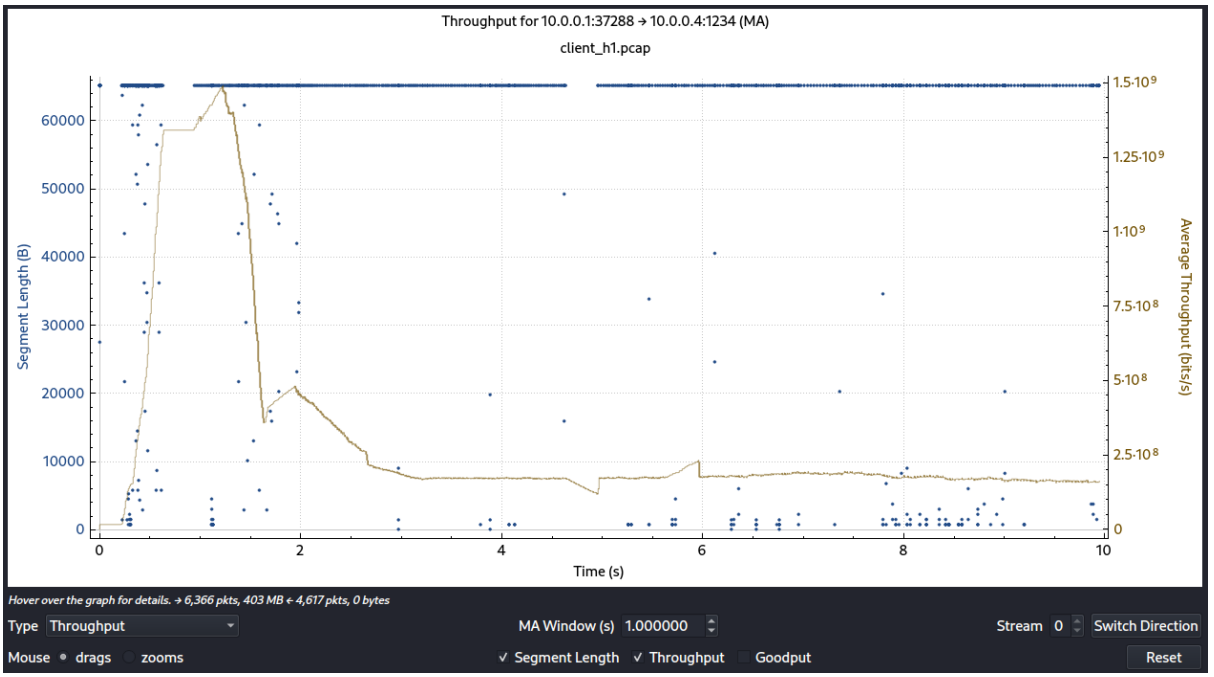


Again it is observed that the curve for throughput of H4 received from H3 is different from that from H2 and H1.

- Congestion Control Scheme: **TCP BBR**

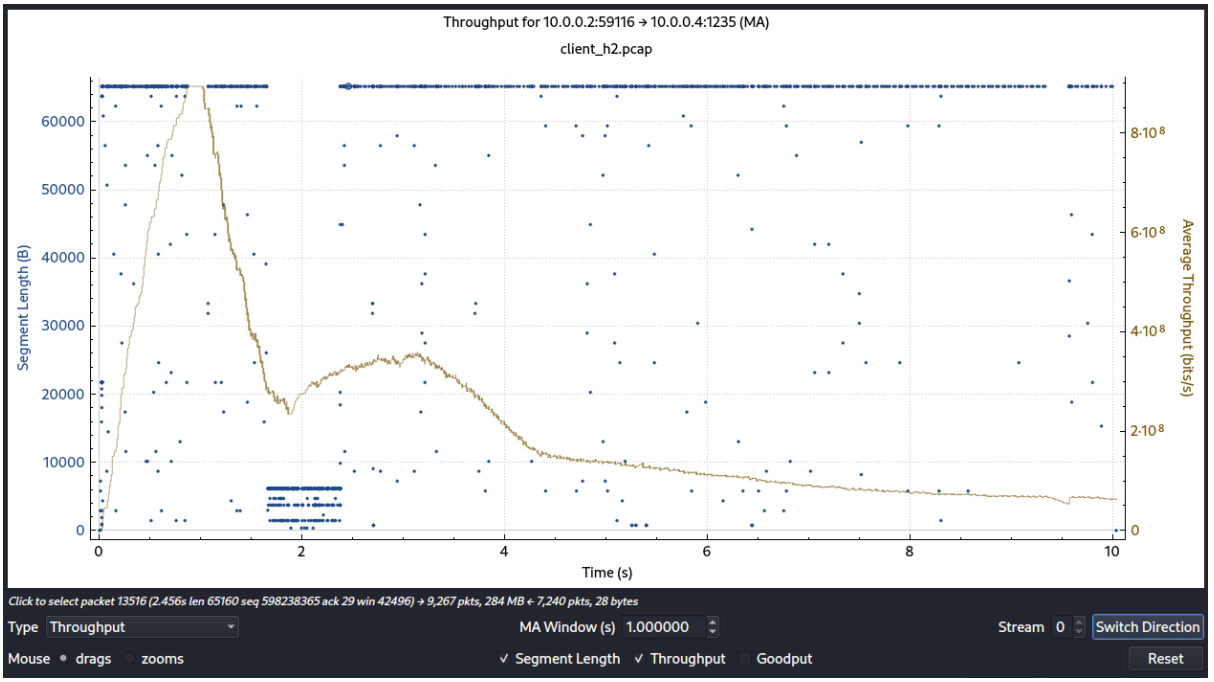
```
Enter configuration (b/c): c
Enter congestion control scheme: bbr
Enter link loss percentage: 0
*** Creating network
*** Adding controller
*** Adding hosts:
h1 h2 h3 h4
*** Adding switches:
s1 s2
*** Adding links:
(s1, h1) (s1, h2) (s1, s2) (s2, h3) (s2, h4)
*** Configuring hosts
h1 h2 h3 h4
*** Starting controller
c0
*** Starting 2 switches
s1 s2 ...
*** Starting CLI:
mininet>
```

For Client H1:

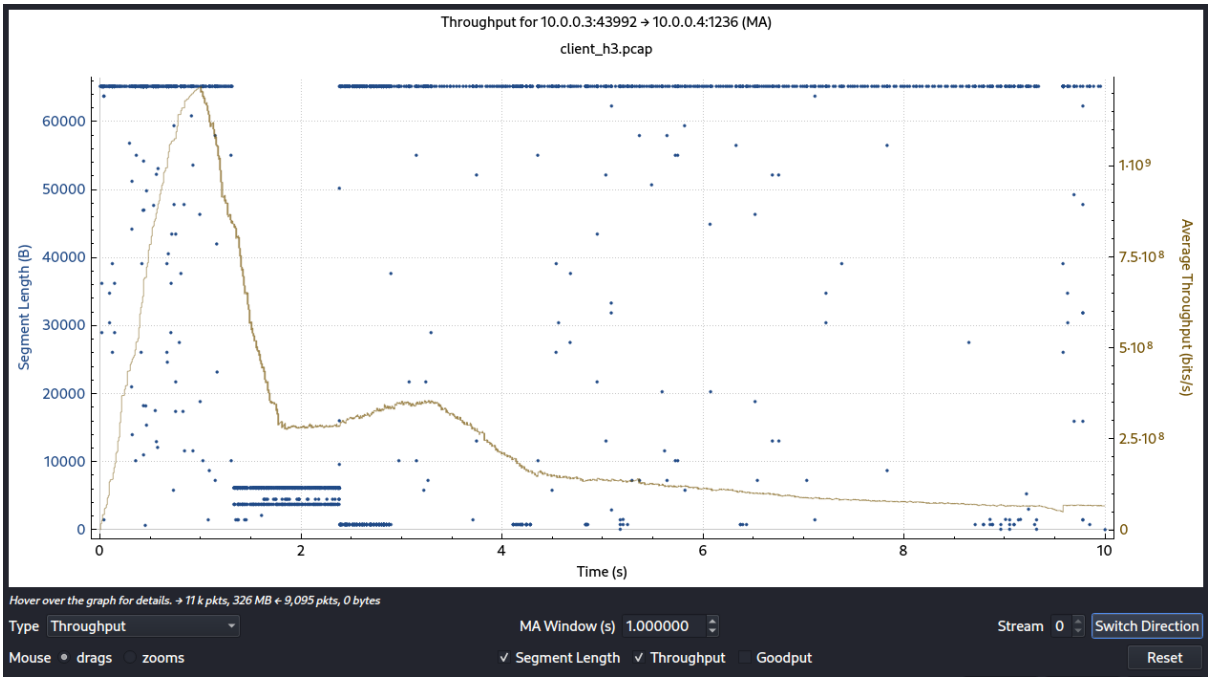


Here, after congestion, the throughput decreases rapidly, then starts increasing, and then again decreases after some period of time. Then it remains somewhat stable at one throughput.

For Client H2:

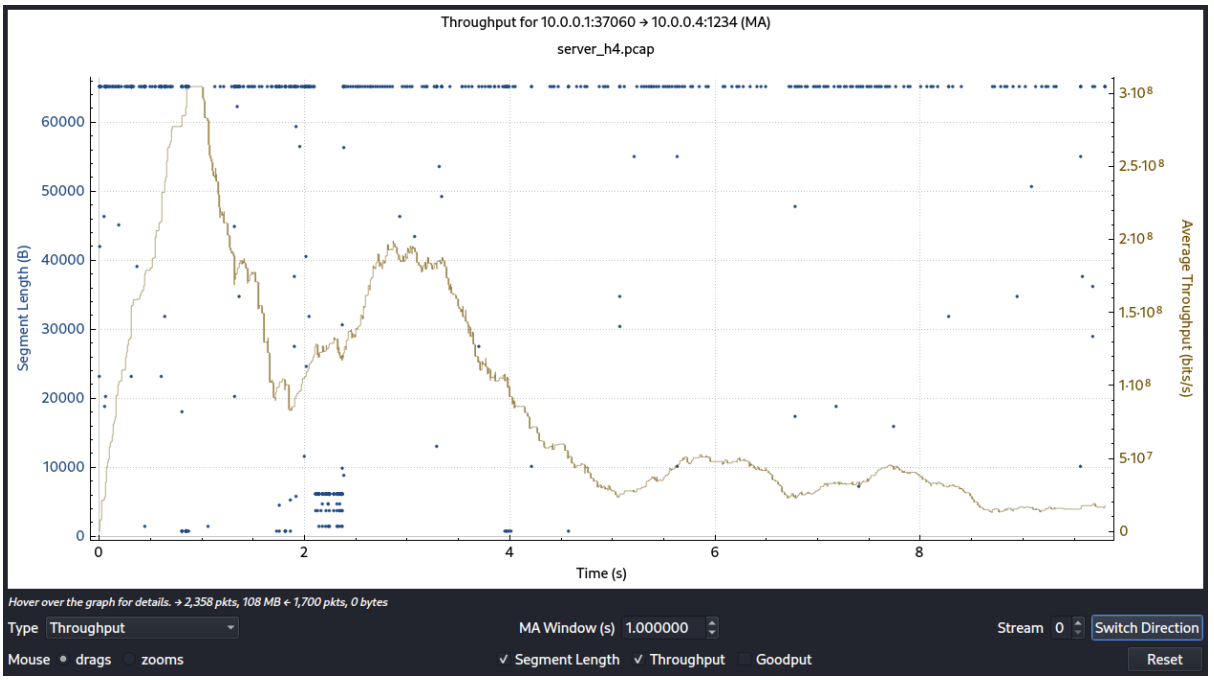


For Client H3:

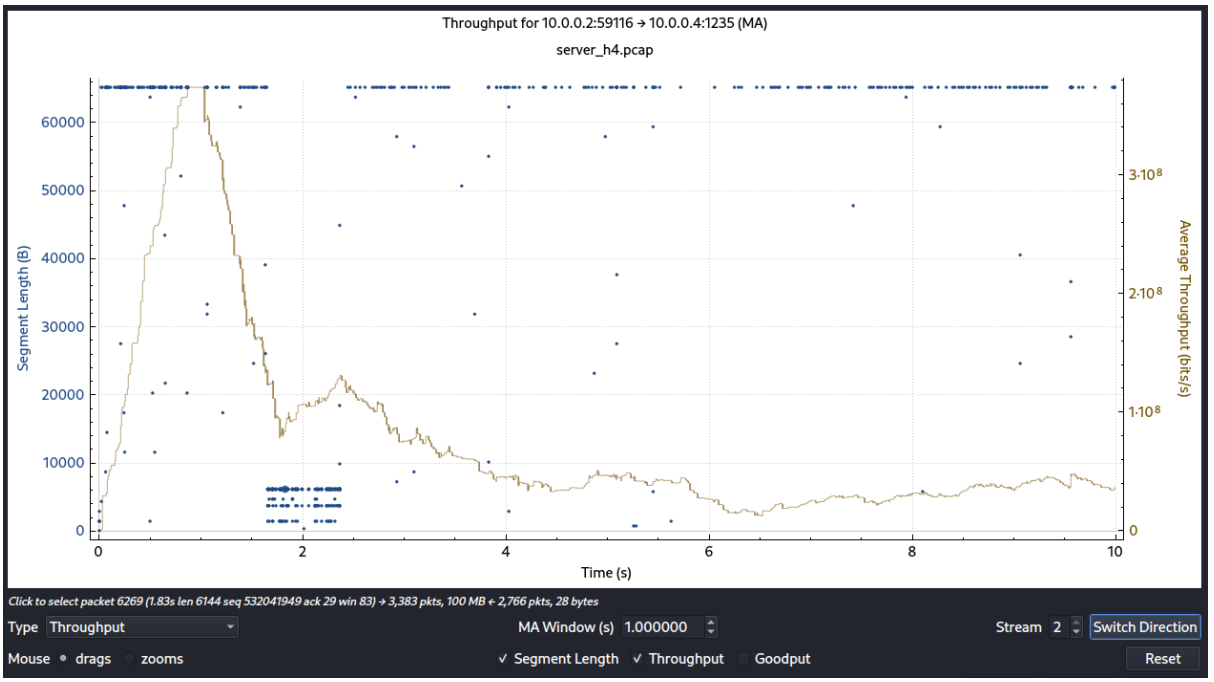


For Server H4:

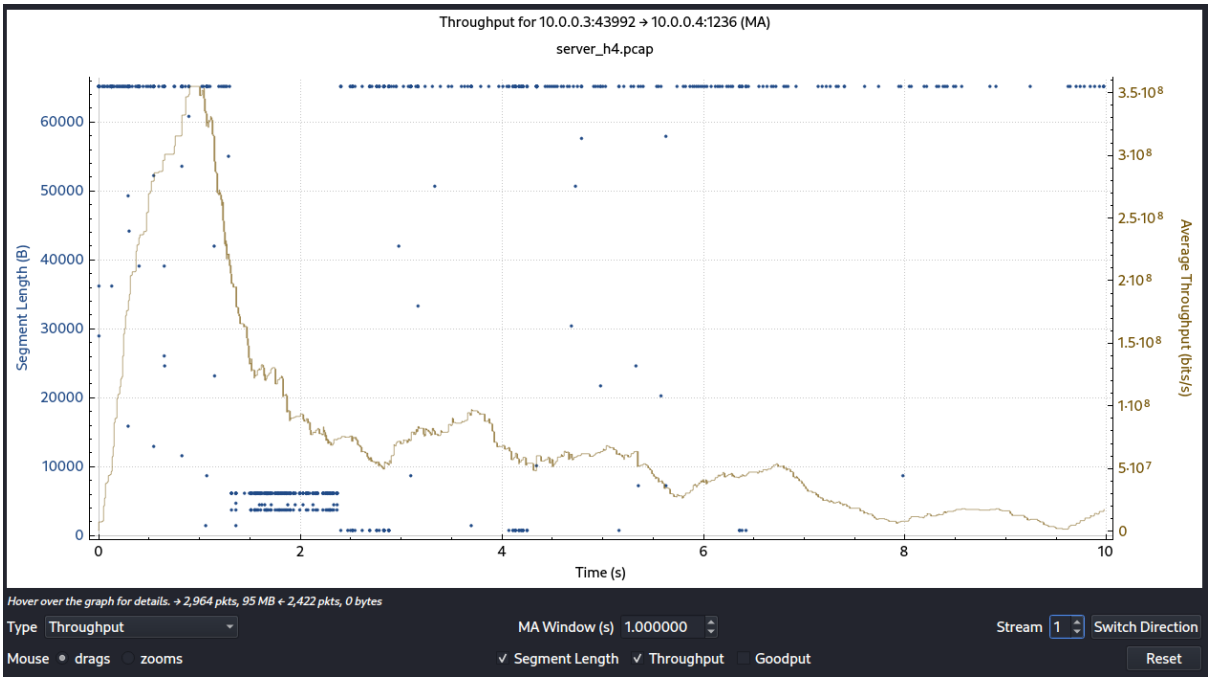
From H1 to H4:



From H2 to H4:



From H3 to H4:


Throughput for 10.0.0.3:43992 → 10.0.0.4:1236 (MA)
server_h4.pcap
Segment Length (B)
Average Throughput (bits/s)
Time (s)
Hover over the graph for details. → 2,964 pkts, 95 MB ← 2,422 pkts, 0 bytes
Type Throughput
MA Window (s) 1.000000
Stream 1
Switch Direction
Mouse drags zooms
Segment Length
Throughput
Goodput
Reset

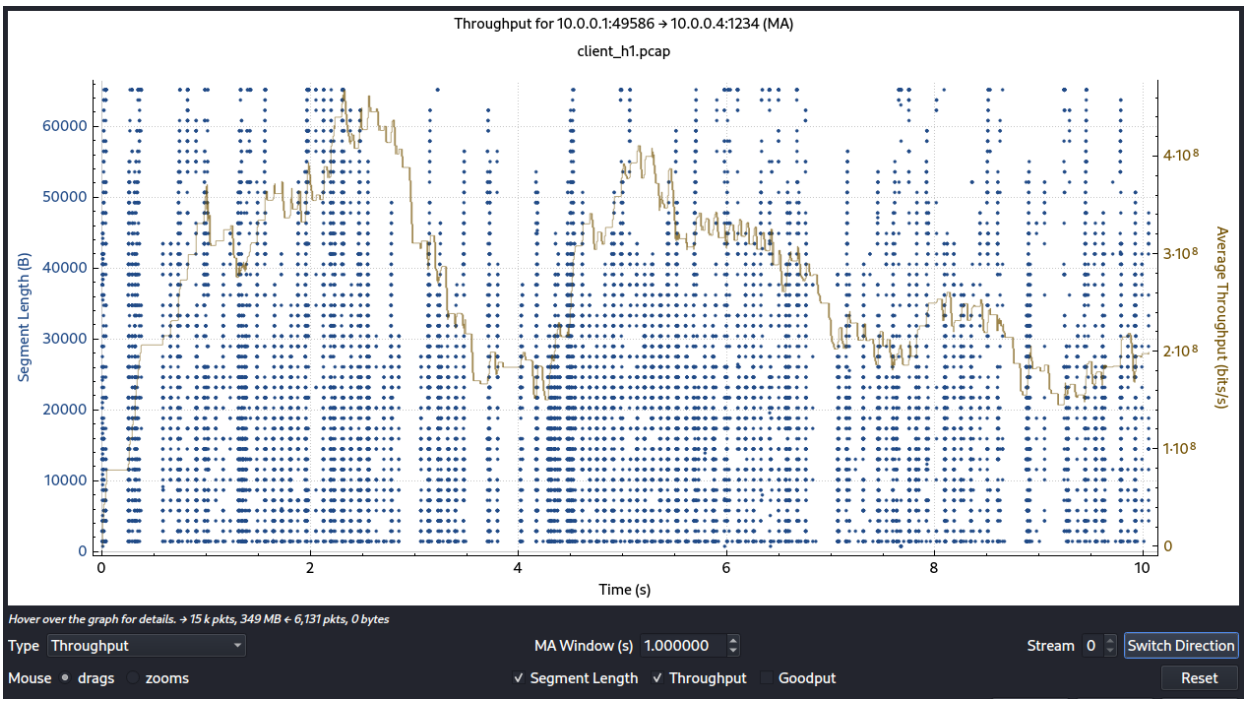
d. **Configure the link loss parameter of the middle link (s1 - s2) to 1% and 3% and run the experiment in (b). Report and compare the throughput over time for each congestion control scheme.**

→

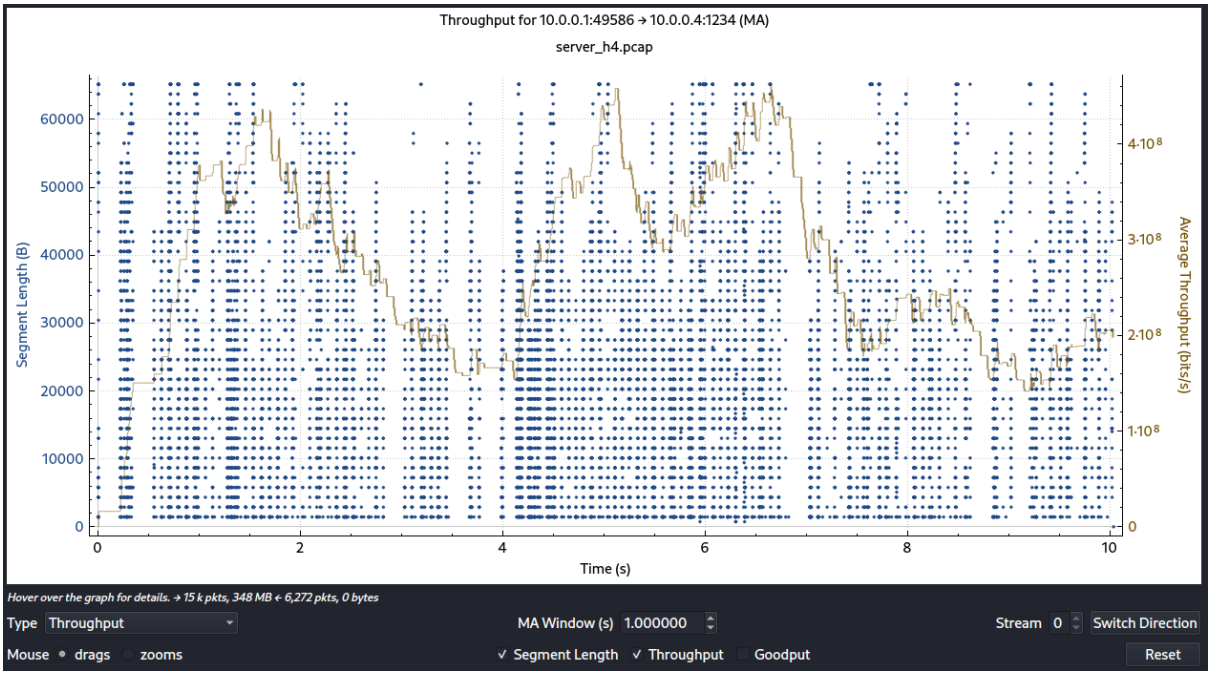
- Congestion Control Scheme: **TCP Vegas**
  Link Loss: 1%:

```
Enter configuration (b/c): b
Enter congestion control scheme: vegas
Enter link loss percentage: 1
*** Creating network
*** Adding controller
*** Adding hosts:
h1 h2 h3 h4
*** Adding switches:
s1 s2
*** Adding links:
(s1, h1) (s1, h2) (1.00000% loss) (1.00000% loss) (s1, s2) (s2, h3) (s2, h4)
*** Configuring hosts
h1 h2 h3 h4
*** Starting controller
c0
*** Starting 2 switches
s1 s2 ...(1.00000% loss) (1.00000% loss)
*** Starting CLI:
mininet>
```

For Client H1:



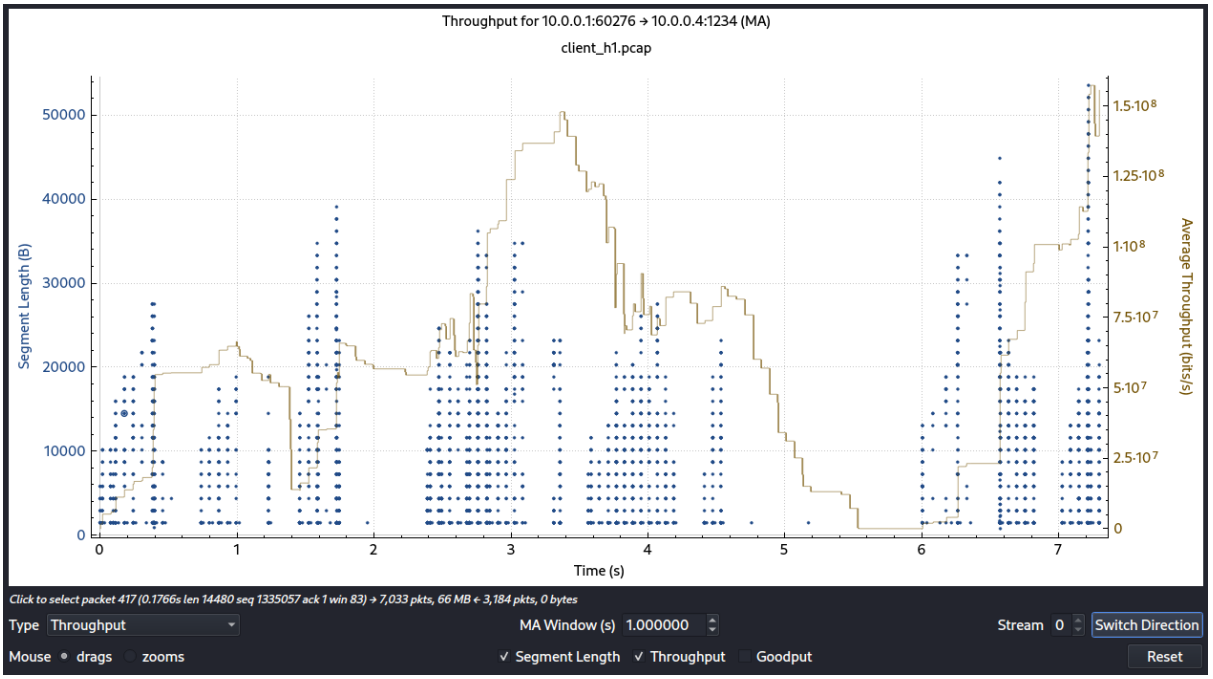
For Server H4:



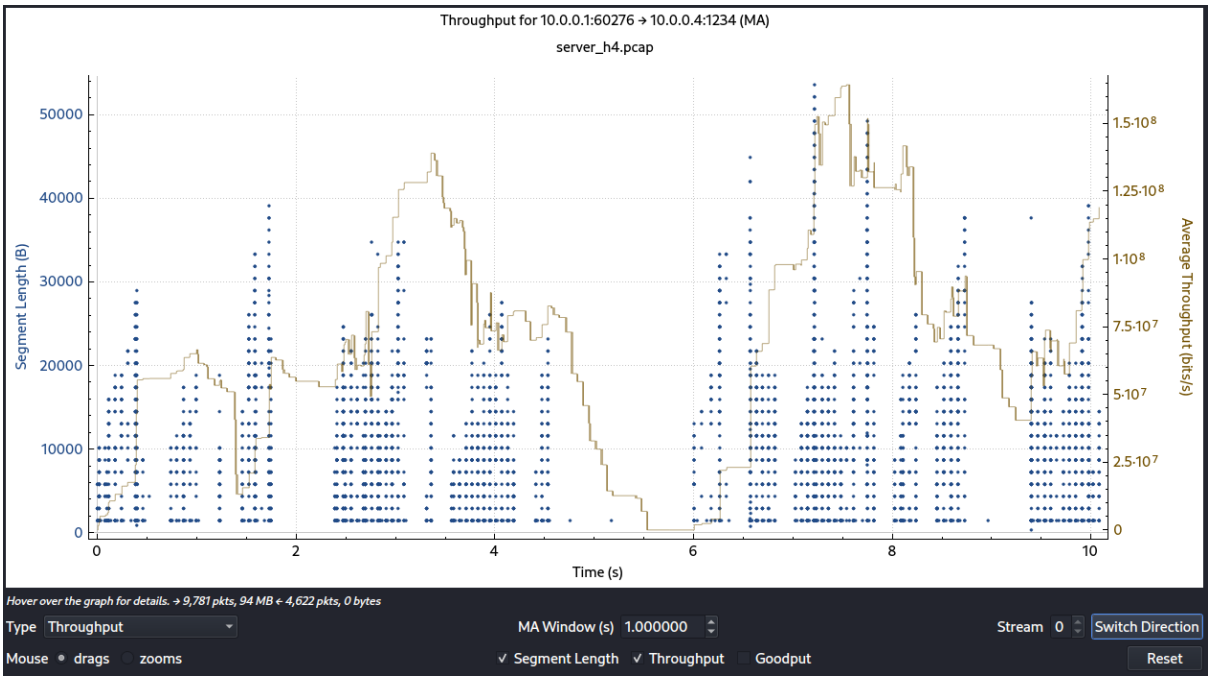
Link Loss: 3%:

```
Enter configuration (b/c): b
Enter congestion control scheme: vegas
Enter link loss percentage: 3
*** Creating network
*** Adding controller
*** Adding hosts:
h1 h2 h3 h4
*** Adding switches:
s1 s2
*** Adding links:
(s1, h1) (s1, h2) (3.00000% loss) (3.00000% loss) (s1, s2) (s2, h3) (s2, h4)
*** Configuring hosts
h1 h2 h3 h4
*** Starting controller
c0
*** Starting 2 switches
s1 s2 ...(3.00000% loss) (3.00000% loss)
*** Starting CLI:
```

For Client H1:



For Server H4:

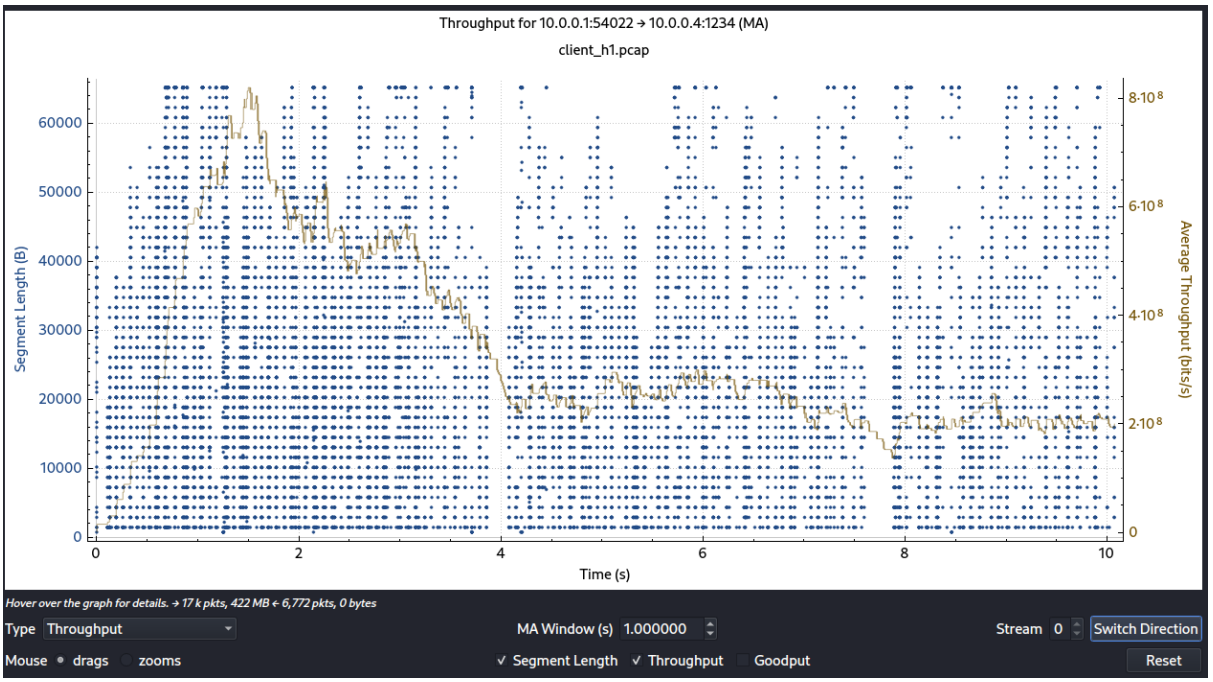


- Congestion Control Scheme: **TCP Reno**

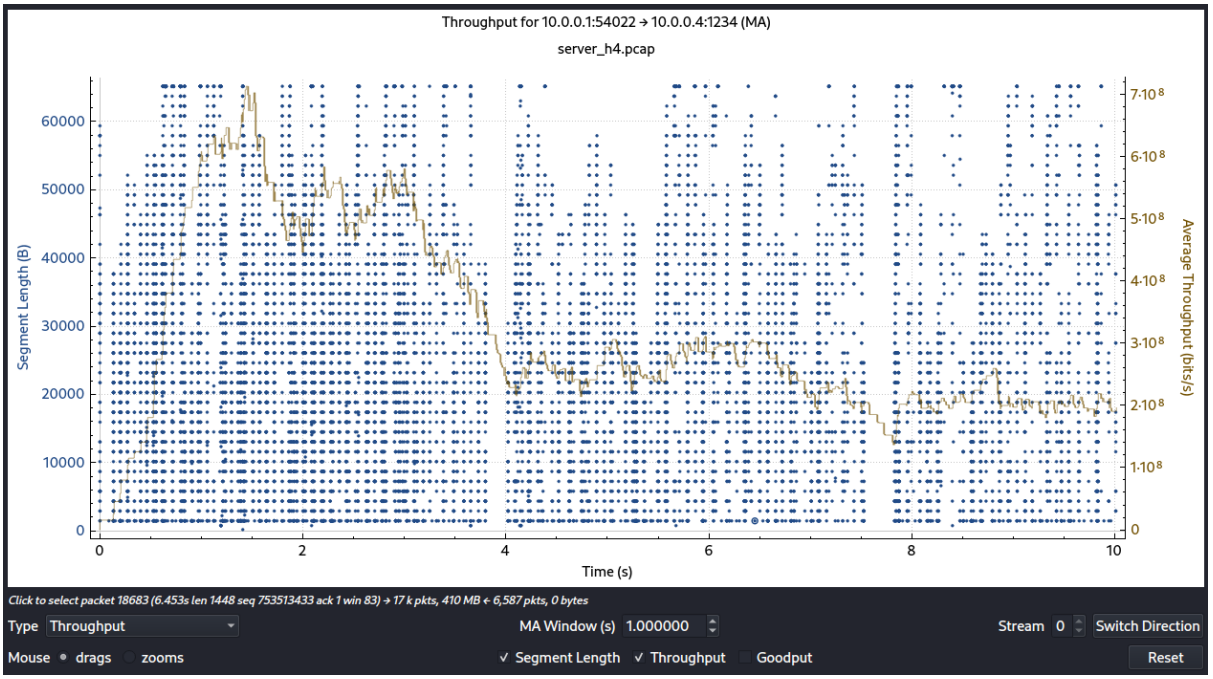
  Link Loss: 1%:

```
Enter configuration (b/c): b
Enter congestion control scheme: reno
Enter link loss percentage: 1
*** Creating network
*** Adding controller
*** Adding hosts:
h1 h2 h3 h4
*** Adding switches:
s1 s2
*** Adding links:
(s1, h1) (s1, h2) (1.00000% loss) (1.00000% loss) (s1, s2) (s2, h3) (s2, h4)
*** Configuring hosts
h1 h2 h3 h4
*** Starting controller
c0
*** Starting 2 switches
s1 s2 ...(1.00000% loss) (1.00000% loss)
*** Starting CLI:
mininet>
```

For Client H1:



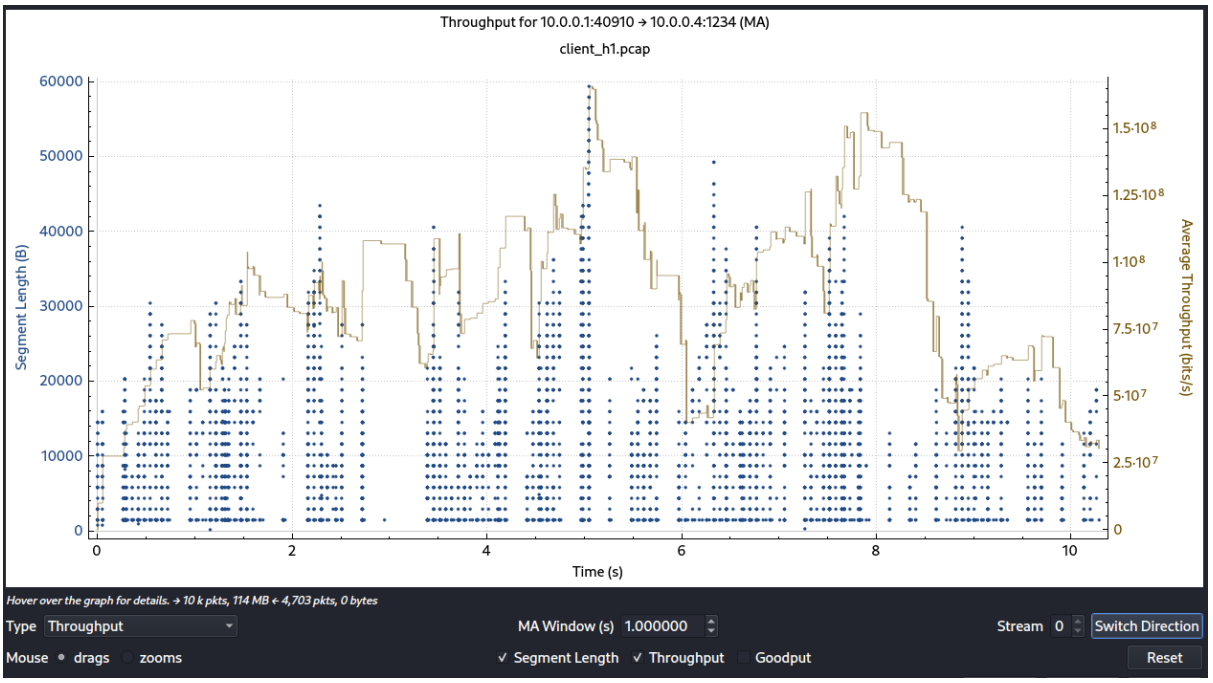
For Server H4:



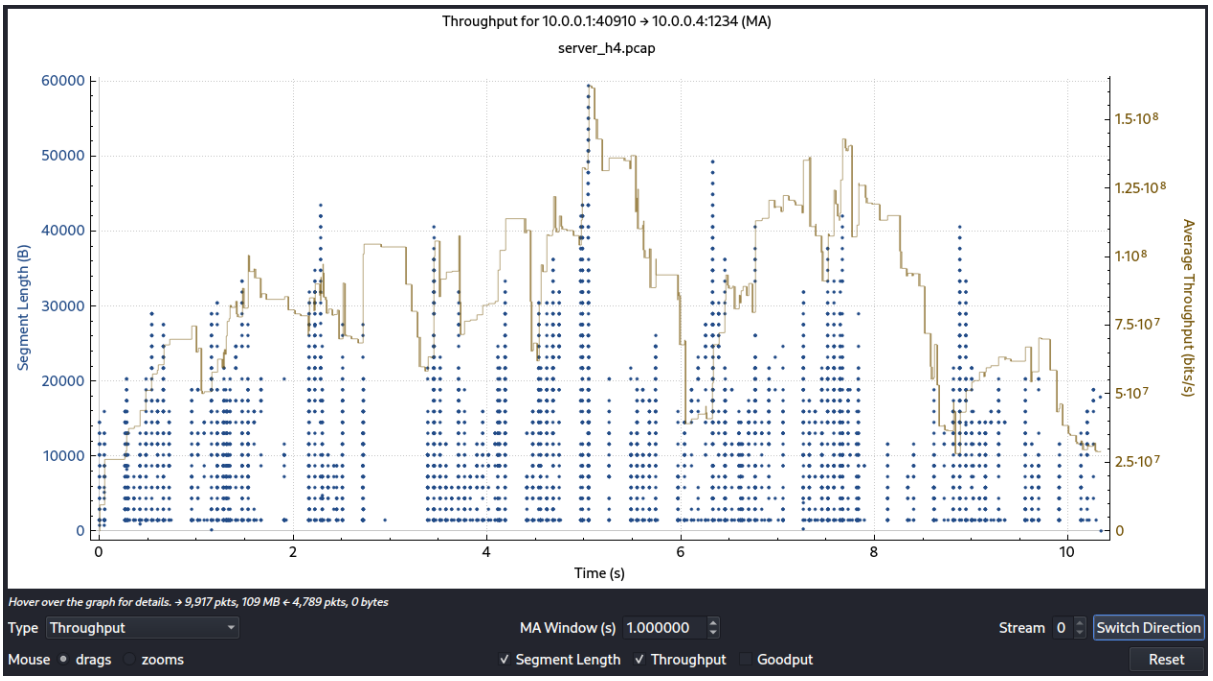
Link Loss: 3%:

```
Enter configuration (b/c): b
Enter congestion control scheme: reno
Enter link loss percentage: 3
*** Creating network
*** Adding controller
*** Adding hosts:
h1 h2 h3 h4
*** Adding switches:
s1 s2
*** Adding links:
(s1, h1) (s1, h2) (3.00000% loss) (3.00000% loss) (s1, s2) (s2, h3) (s2, h4)
*** Configuring hosts
h1 h2 h3 h4
*** Starting controller
c0
*** Starting 2 switches
s1 s2 ...(3.00000% loss) (3.00000% loss)
*** Starting CLI:
mininet>
```

For Client H1:



For Server H4:

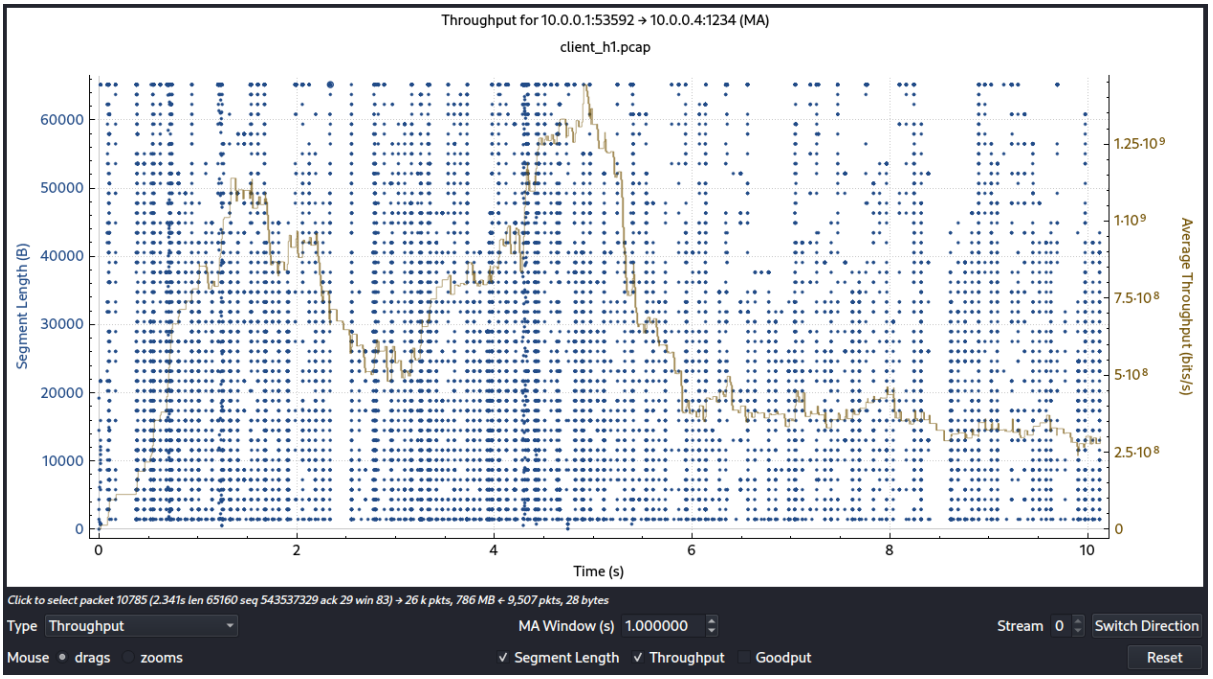


- Congestion Control Scheme: **TCP Cubic**

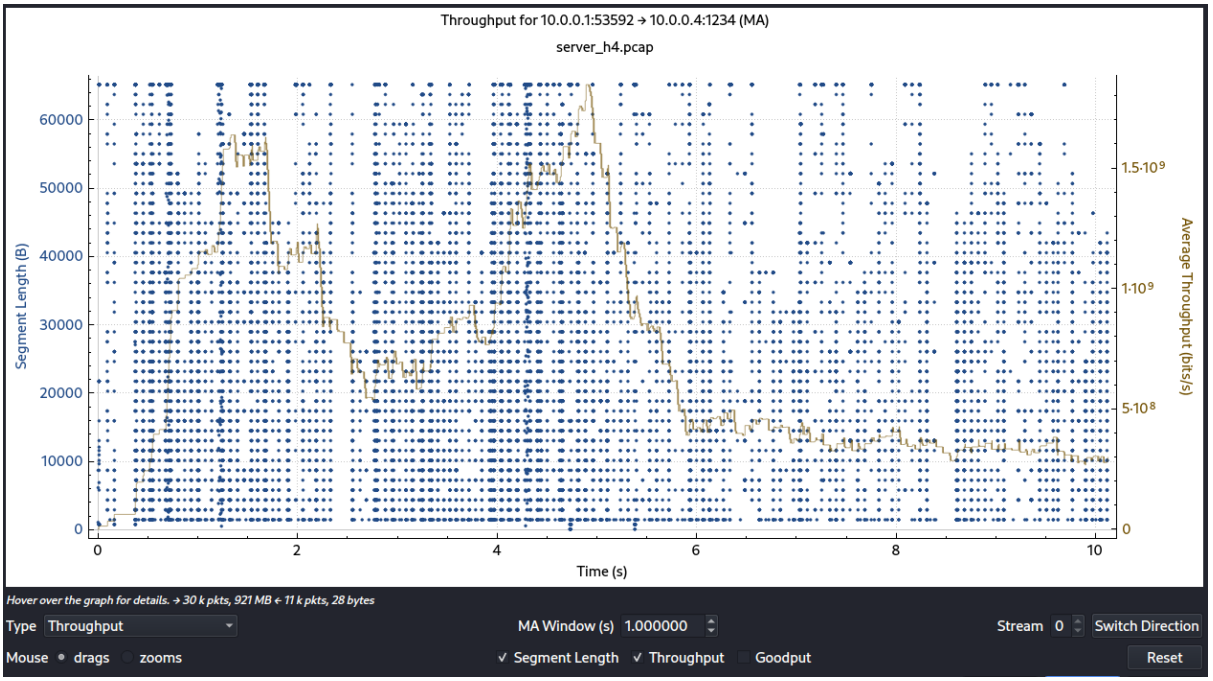
  Link Loss: 1%:

```
Enter configuration (b/c): b
Enter congestion control scheme: cubic
Enter link loss percentage: 1
*** Creating network
*** Adding controller
*** Adding hosts:
h1 h2 h3 h4
*** Adding switches:
s1 s2
*** Adding links:
(s1, h1) (s1, h2) (1.00000% loss) (1.00000% loss) (s1, s2) (s2, h3) (s2, h4)
*** Configuring hosts
h1 h2 h3 h4
*** Starting controller
c0
*** Starting 2 switches
s1 s2 ...(1.00000% loss) (1.00000% loss)
*** Starting CLI:
mininet>
```

For Client H1:



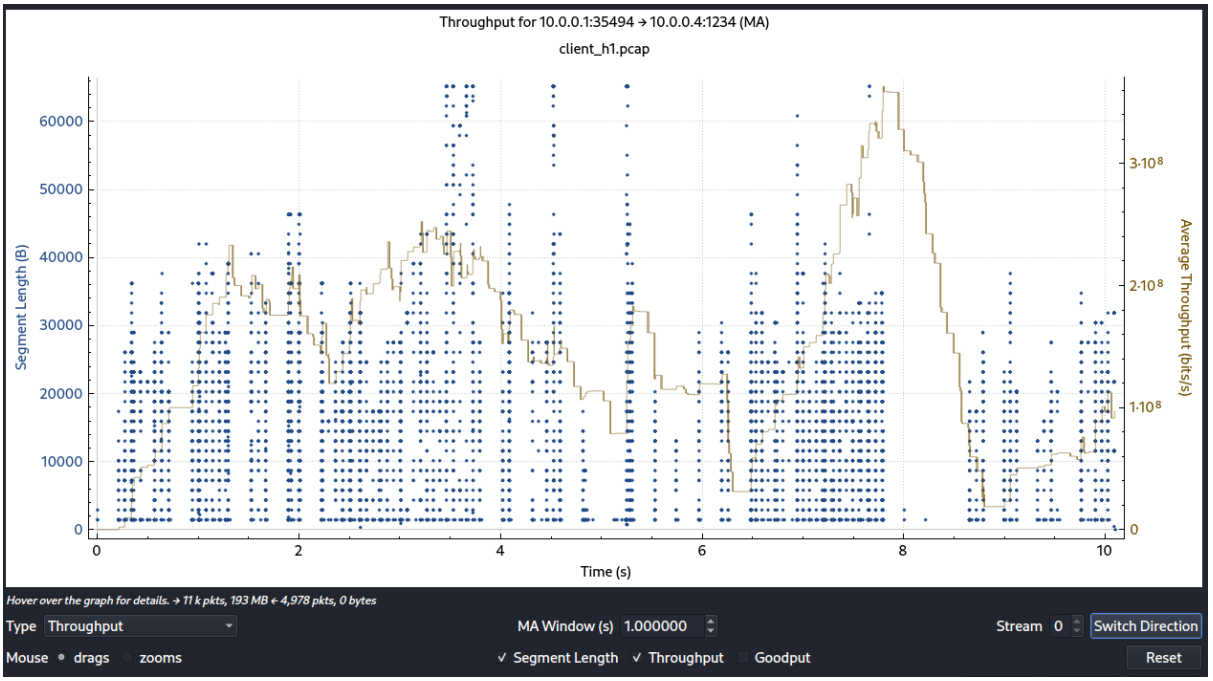
For Server H4:



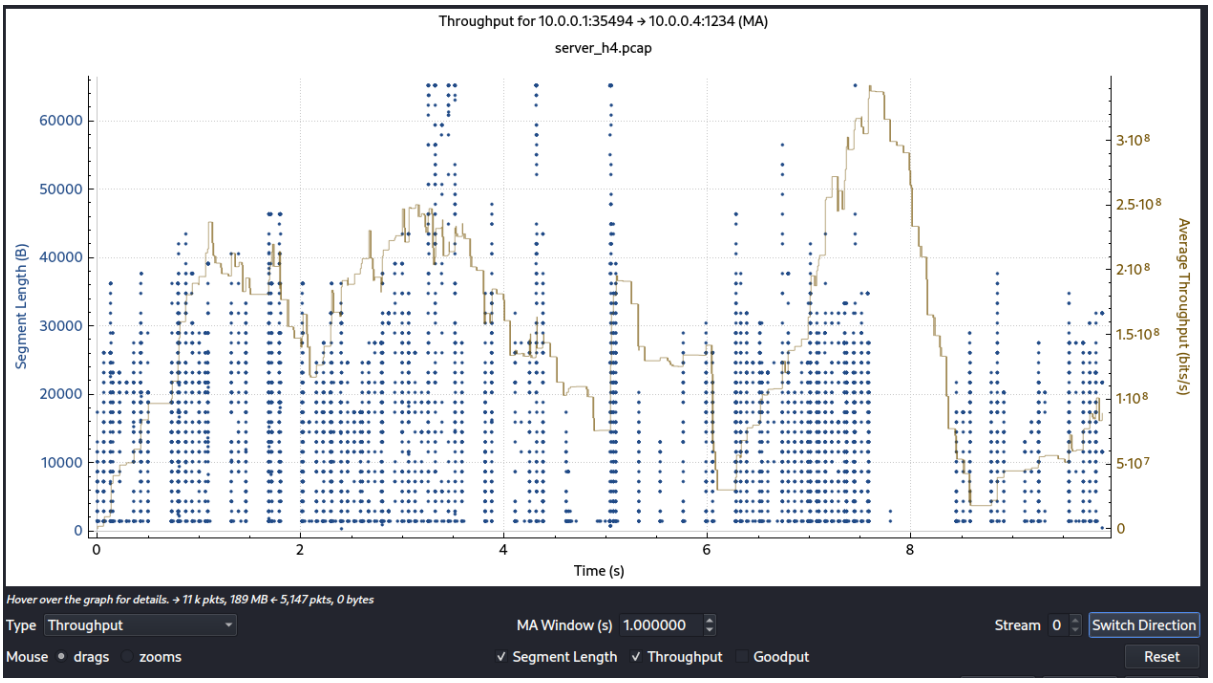
Link Loss: 3%:

```
Enter configuration (b/c): b
Enter congestion control scheme: cubic
Enter link loss percentage: 3
*** Creating network
*** Adding controller
*** Adding hosts:
h1 h2 h3 h4
*** Adding switches:
s1 s2
*** Adding links:
(s1, h1) (s1, h2) (3.00000% loss) (3.00000% loss) (s1, s2) (s2, h3) (s2, h4)
*** Configuring hosts
h1 h2 h3 h4
*** Starting controller
c0
*** Starting 2 switches
s1 s2 ...(3.00000% loss) (3.00000% loss)
*** Starting CLI:
```

For Client H1:



For Server H4:

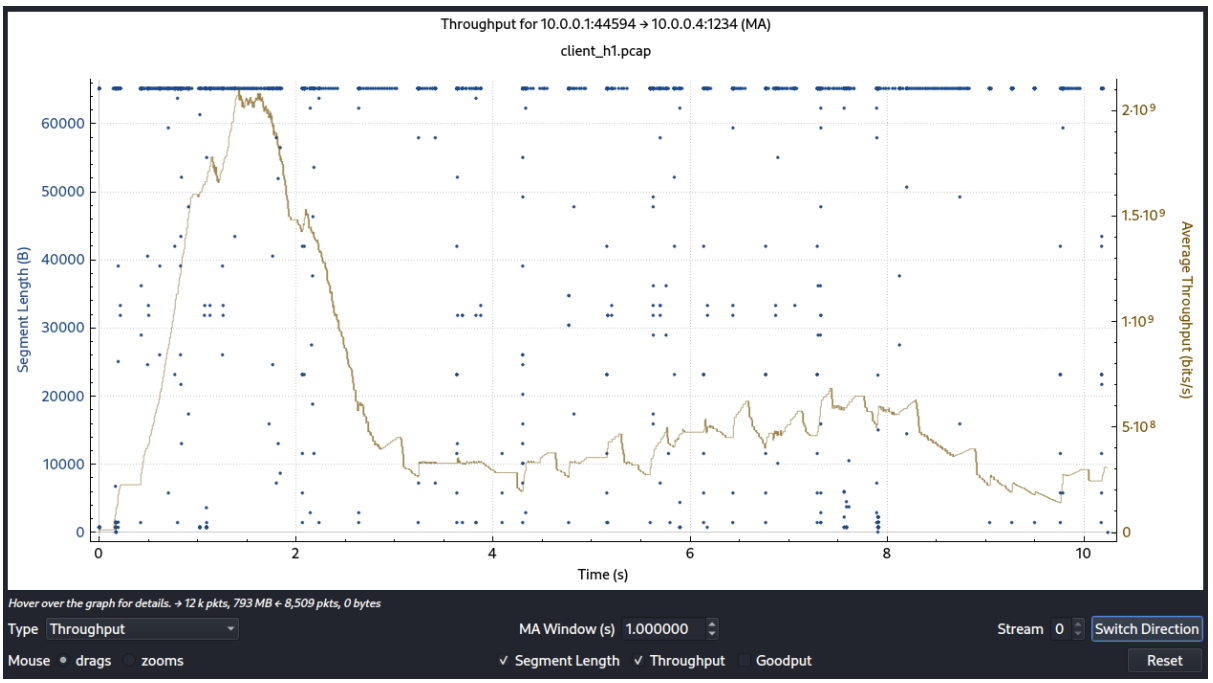


- Congestion Control Scheme: **TCP BBR**

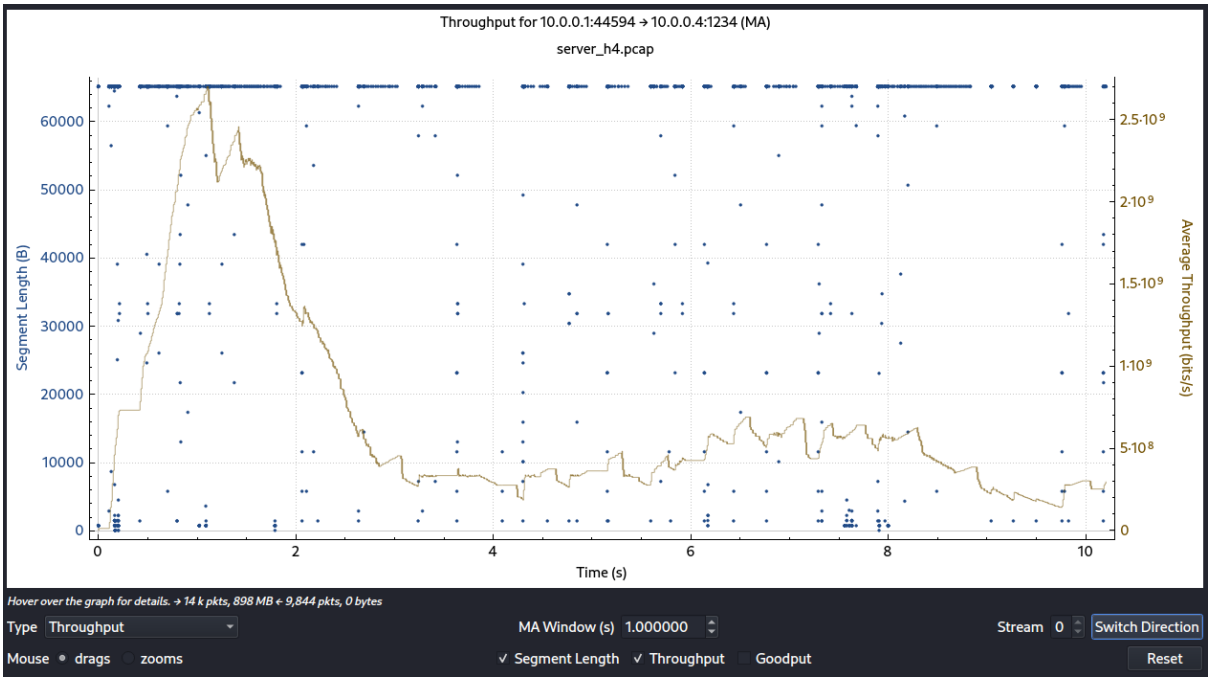
  Link Loss: 1%:

```
Enter configuration (b/c): b
Enter congestion control scheme: bbr
Enter link loss percentage: 1
*** Creating network
*** Adding controller
*** Adding hosts:
h1 h2 h3 h4
*** Adding switches:
s1 s2
*** Adding links:
(s1, h1) (s1, h2) (1.00000% loss) (1.00000% loss) (s1, s2) (s2, h3) (s2, h4)
*** Configuring hosts
h1 h2 h3 h4
*** Starting controller
c0
*** Starting 2 switches
s1 s2 ...(1.00000% loss) (1.00000% loss)
*** Starting CLI:
mininet>
```

For Client H1:



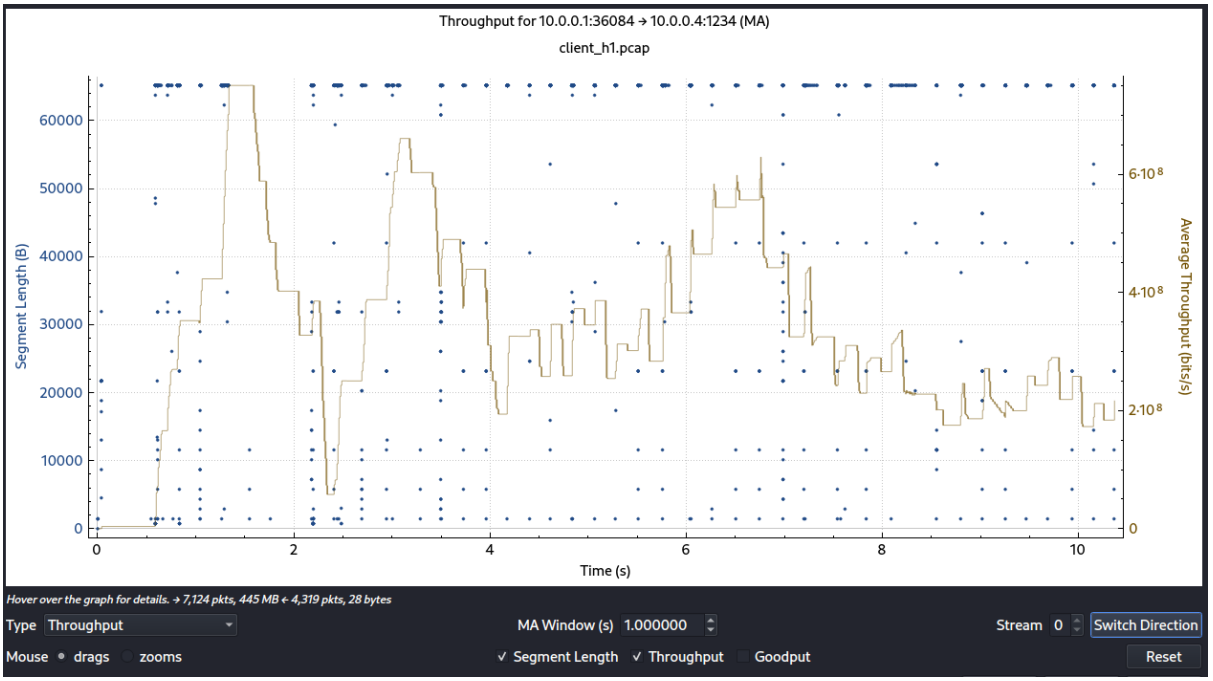
For Server H4:



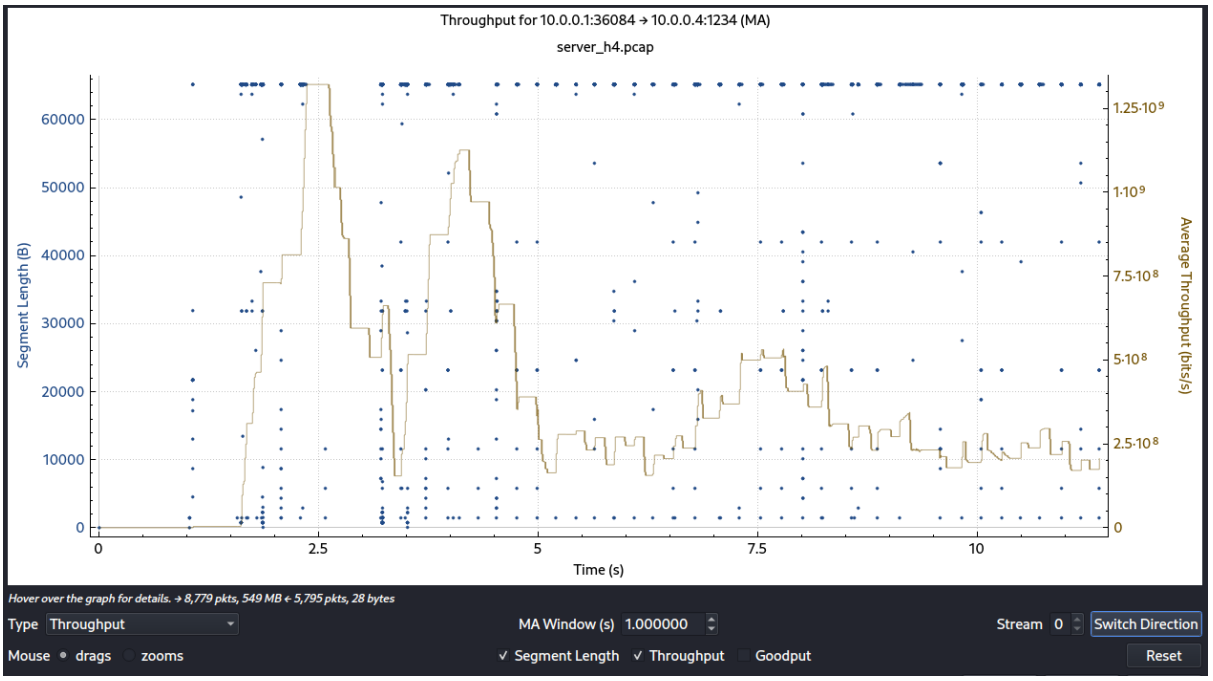
Link Loss: 3%:

```
Enter configuration (b/c): b
Enter congestion control scheme: bbr
Enter link loss percentage: 3
*** Creating network
*** Adding controller
*** Adding hosts:
h1 h2 h3 h4
*** Adding switches:
s1 s2
*** Adding links:
(s1, h1) (s1, h2) (3.00000% loss) (3.00000% loss) (s1, s2) (s2, h3) (s2, h4)
*** Configuring hosts
h1 h2 h3 h4
*** Starting controller
c0
*** Starting 2 switches
s1 s2 ...(3.00000% loss) (3.00000% loss)
*** Starting CLI:
```

For Client H1:



For Server H4:



Overall Comparison:

We can see that as an overview that the throughput when 1% link loss, is atmost 3 times the throughput when the loss is 3%

References:

- https://atoonk.medium.com/tcp-bbr-exploring-tcp-congestion-control-84c9c11dc3a9
- https://iperf.fr/iperf-doc.php
- https://github.com/mininet/mininet/blob/master/examples/linuxrouter.py
- https://cloudcone.com/docs/article/how-to-test-network-throughput-using-iperf3-tool/
- https://webcms3.cse.unsw.edu.au/static/uploads/course/COMP3331/16s1/894894253a9d7bb9b3575af5092c2d80c9382bbbf860e4a9364cfae2bcf04cd6/Lab3a.pdf
- http://mininet.org/walkthrough/
- https://www.youtube.com/watch?v=yHUNeyaQKWY
- https://aneescraftsmanship.com/mininet-default-topology-example/
- https://stackoverflow.com/questions/46595423/mininet-how-to-create-a-topology-with-two-routers-and-their-respective-hosts
- https://docs.python.org/3/library/cmd.html
- https://www.ateam-oracle.com/post/testing-latency-and-throughput
- https://www.scribd.com/document/305348776/Topology-in-mininet