ASSIGNMENT - II

Team - Sachin Jalan (21110183) Anushk Bhana (21110031)

Answer 1

In the first question we had to implement the topology given below in mininet using python api:

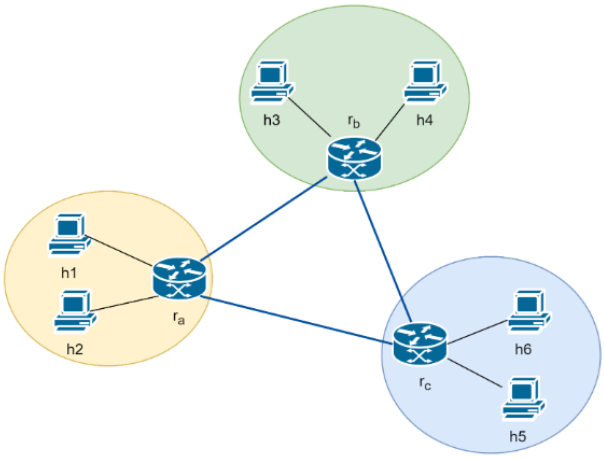

Fig 1.1: Network Topology

We used the router code given in the examples and configured the topology. We used the cmd command to put the entries in the routing tables of the respective routers. This led to the understanding of subnets and routing table. We have used static routes for the question.

For the part 1 of the question where we have to show complete connectivity:

```
└─$ sudo python3 topop.py
*** Creating network
*** Adding controller
*** Adding hosts:
d1 d2 d3 d4 d5 d6 r1 r2 r3
*** Adding switches:
s1 s2 s3
*** Adding links:
(r1, r2) (r1, r3) (r2, r3) (s1, d1) (s1, d4) (s1, r1) (s2, d2) (s2, d5) (s2,
r2) (s3, d3) (s3, d6) (s3, r3)
*** Configuring hosts
d1 d2 d3 d4 d5 d6 r1 r2 r3
*** Starting controller
c0
*** Starting 3 switches
s1 s2 s3 ...
*** Starting CLI:
mininet> pingall
*** Ping: testing ping reachability
d1 -> d2 d3 d4 d5 d6 r1 r2 r3
d2 -> d1 d3 d4 d5 d6 r1 r2 r3
d3 -> d1 d2 d4 d5 d6 r1 r2 r3
d4 -> d1 d2 d3 d5 d6 r1 r2 r3
d5 -> d1 d2 d3 d4 d6 r1 r2 r3
d6 -> d1 d2 d3 d4 d5 r1 r2 r3
r1 -> d1 d2 d3 d4 d5 d6 r2 r3
r2 -> d1 d2 d3 d4 d5 d6 r1 r3
r3 -> d1 d2 d3 d4 d5 d6 r1 r2
*** Results: 0% dropped (72/72 received)
```

Fig 1.2: Mininet Emulation

Fig 1.2 shows the screenshot when the code for the topology was run, it shows when the command pingall is run what happens. As we can see in Fig 1.2 all the hosts d1 to d6 are connected and the routers are also connected.

The configuration of the setup is as follows :

The hosts d1 d4 are connected to router r1 via a switch s1, hosts d2 d5 are connected to router r2 via a switch s2, host d3,d6 are connected to router r3 via a switch s3.

In the part b of the question we had to show the capturing of packets on one of the routers

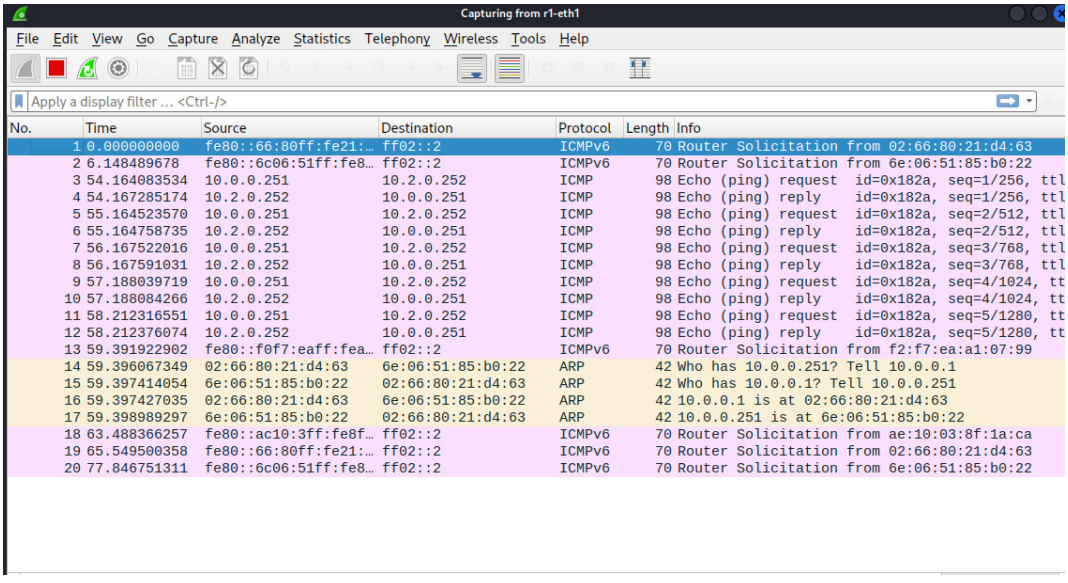

Fig 1.3 : Capturing of packets on wireshark for router 1 interface - eth1

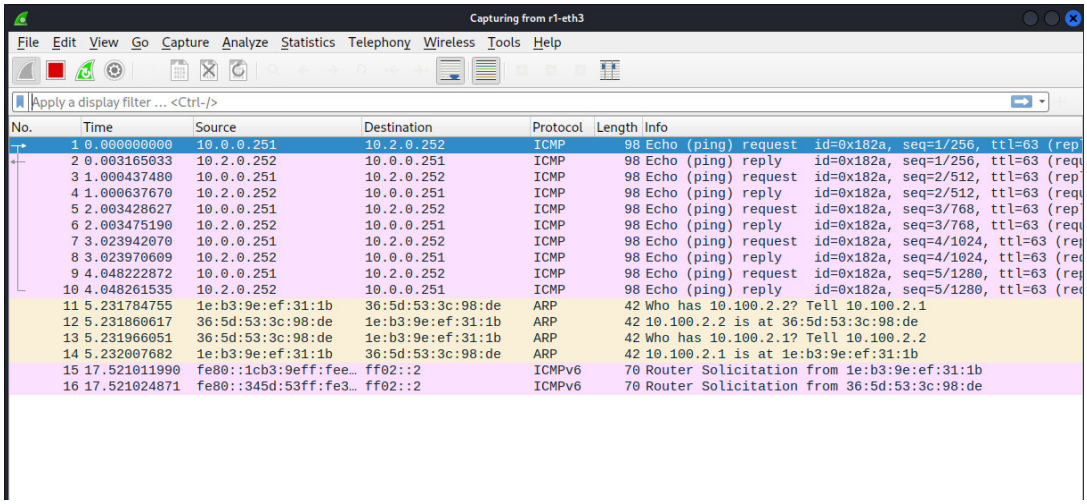

Fig 1.4: Capturing of packets on wireshark for router r1 interface eth3

Figure 1.3 shows the packets that are being sniffed at the interface r1-eth1 which connects to the switch connecting to host 1 and figure 1.4 shows the packets that are being captured on the interface r1-eth3 when d1 is pinging d3, d1 is connected to r1 and d3 to r3.

```
mininet> dump
<Host d1: d1-eth0:10.0.0.251 pid=30690>
<Host d2: d2-eth0:10.1.0.252 pid=30692>
<Host d3: d3-eth0:10.2.0.252 pid=30694>
<Host d4: d4-eth0:10.0.0.145 pid=30696>
<Host d5: d5-eth0:10.1.0.145 pid=30698>
<Host d6: d6-eth0:10.2.0.145 pid=30700>
<LinuxRouter r1: r1-eth1:10.0.0.1,r1-eth2:10.100.0.1,r1-eth3:10.100.2.1 pid=3
0704>
<LinuxRouter r2: r2-eth1:10.1.0.1,r2-eth2:10.100.0.2,r2-eth3:10.100.1.1 pid=3
0706>
<LinuxRouter r3: r3-eth1:10.2.0.1,r3-eth2:10.100.1.2,r3-eth3:10.100.2.2 pid=3
0708>
<OVSSwitch s1: lo:127.0.0.1,s1-eth1:None,s1-eth2:None,s1-eth3:None pid=30713>

<OVSSwitch s2: lo:127.0.0.1,s2-eth1:None,s2-eth2:None,s2-eth3:None pid=30716>

<OVSSwitch s3: lo:127.0.0.1,s3-eth1:None,s3-eth2:None,s3-eth3:None pid=30719>

<OVSController c0: 127.0.0.1:6653 pid=30683>
```

Fig 1.5 : The information of the hosts switches and controllers

In C part of question 1 we had to vary the paths and show the latency difference between the 2 paths.

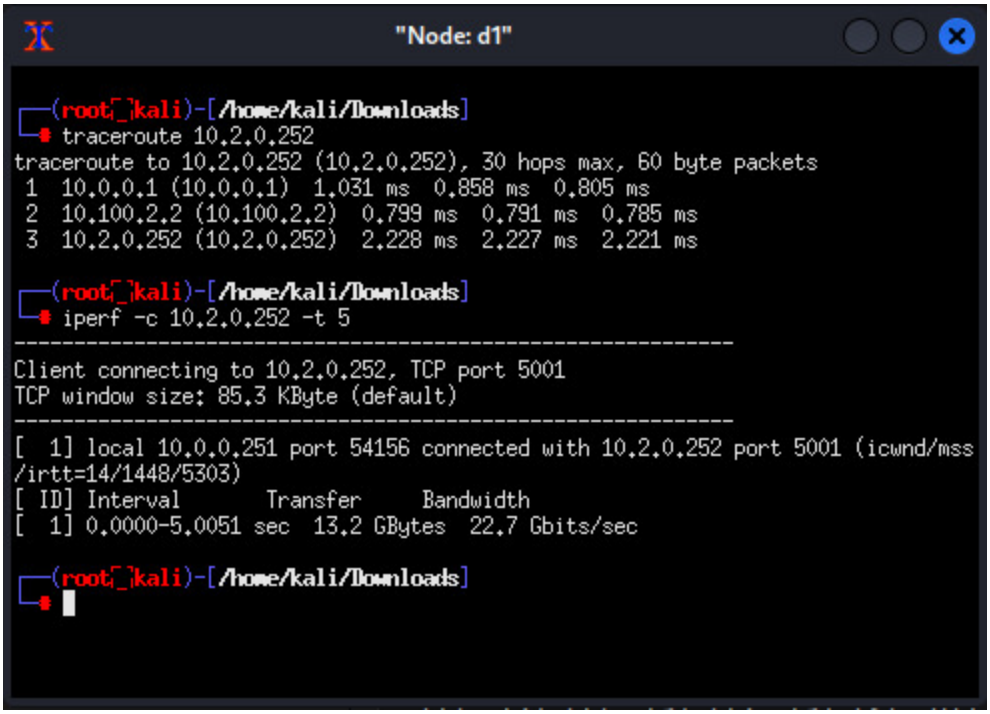

Fig 1.6 : Traceroute and iperf when r1 - r3 are directly connected

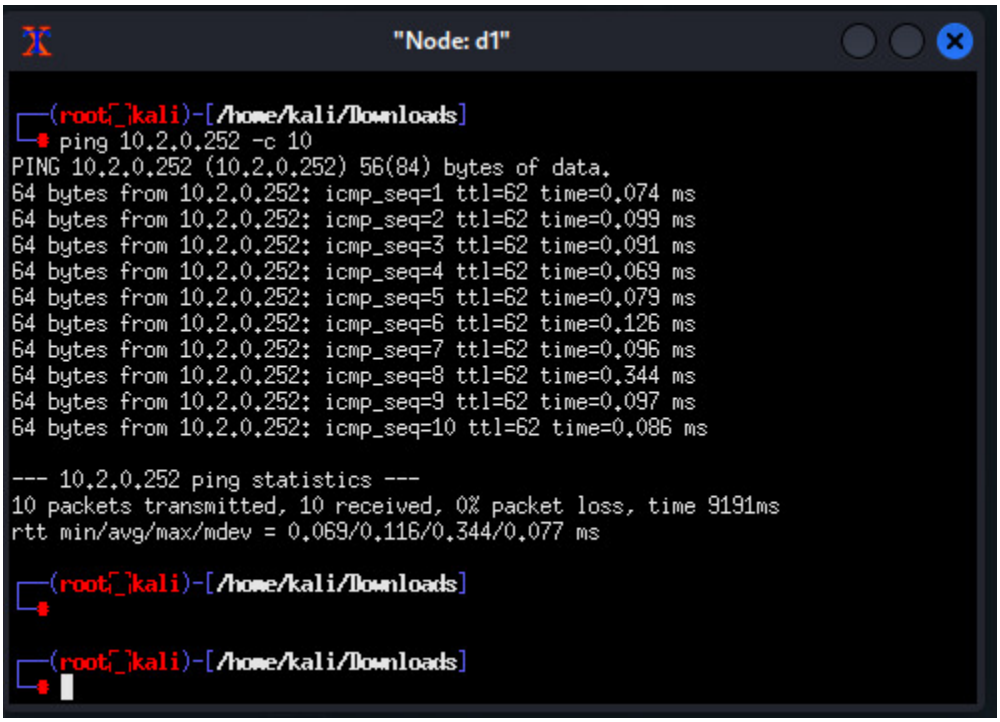

Fig 1.7 : Ping result when r1 - r3 are directly connected

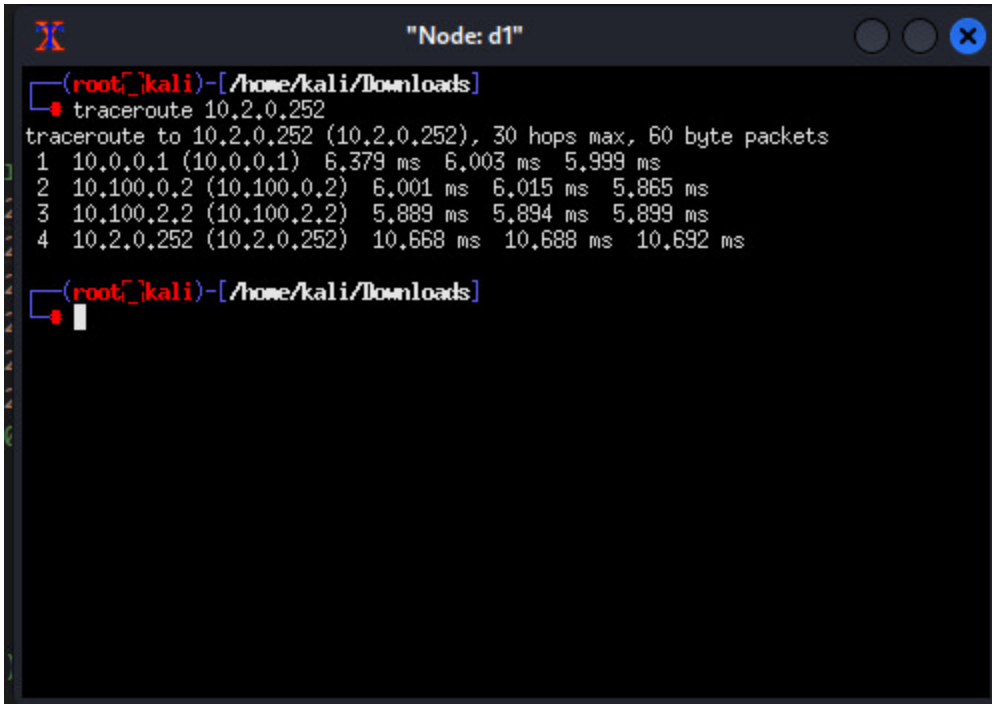

Fig 1.8: Traceroute when r1–r2–r3 is the path for packet

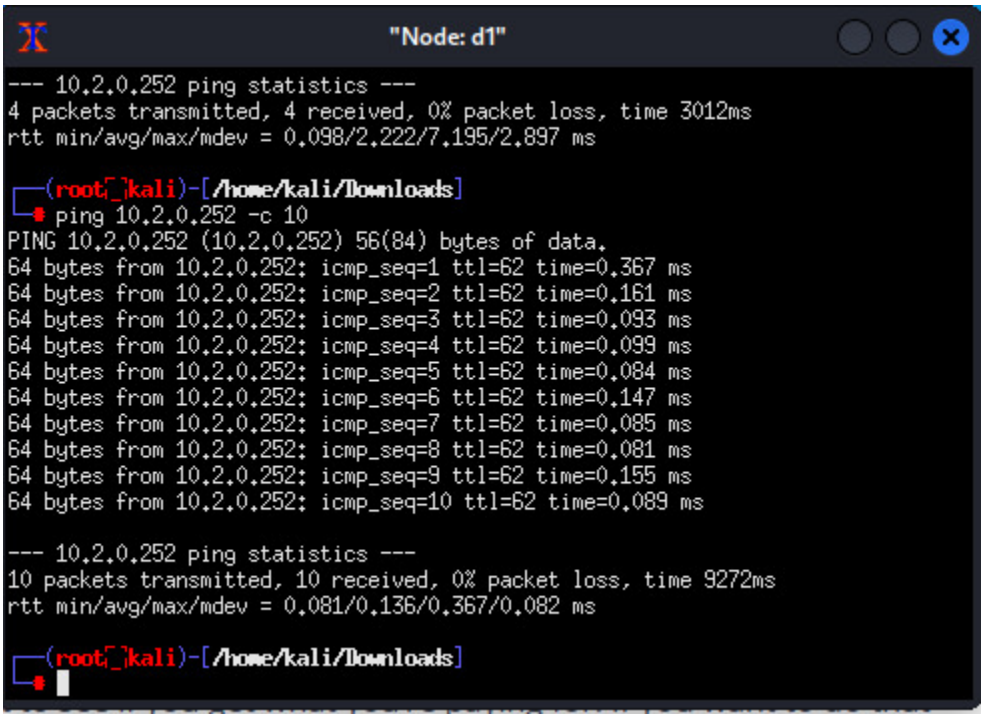

Fig 1.9 : Ping results when r1–r2–r3 is the path

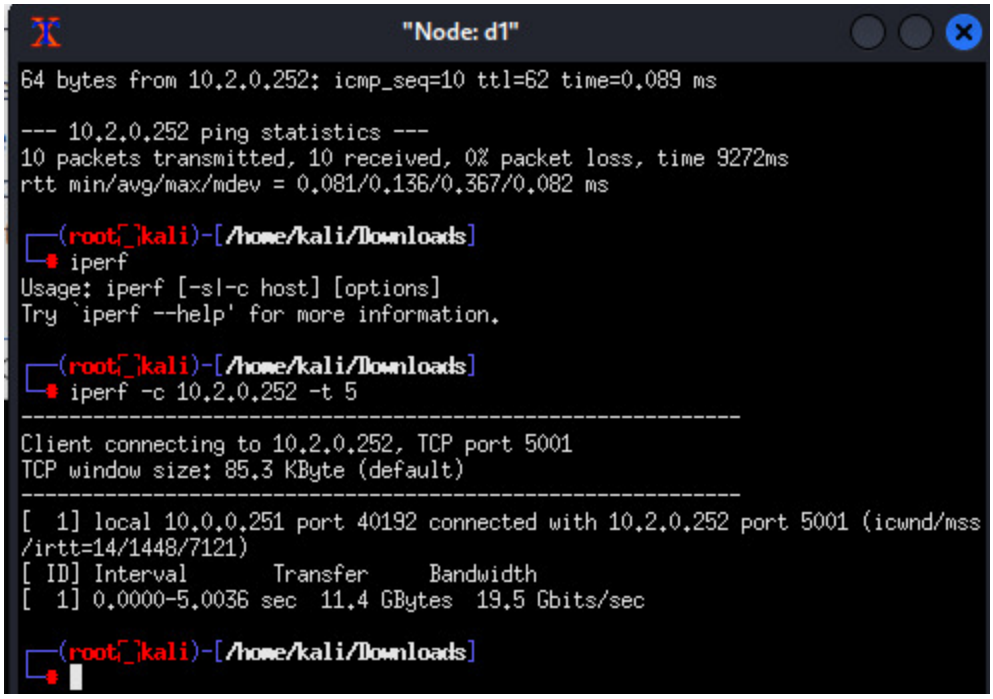

Fig 1.10: Iperf result when r1–r2–r3 is the path

As we can see from the attached screenshot the difference between the rtt and the bandwidth for the packets while varying the path. Fig 1.6 and Fig 1.7 are the screenshots when router 1 and router 2 are directly connected, that means the packet going from the router 1 will go directly to router 3 via r1-r3 link, also it is visible from the traceroute command the path of the packets. The average rtt is 0.116 ms and average bandwidth is 22.7 Gbits/sec in case 1 while for the case when packets arriving at r1 have to go via r2 then r3 the average rtt is 0.136ms and average bandwidth is 19.5 GBits/sec. Hence as the packets have to go through another link we can see the difference in the latency.

In part D we had to dump the routing tables for all the routers :

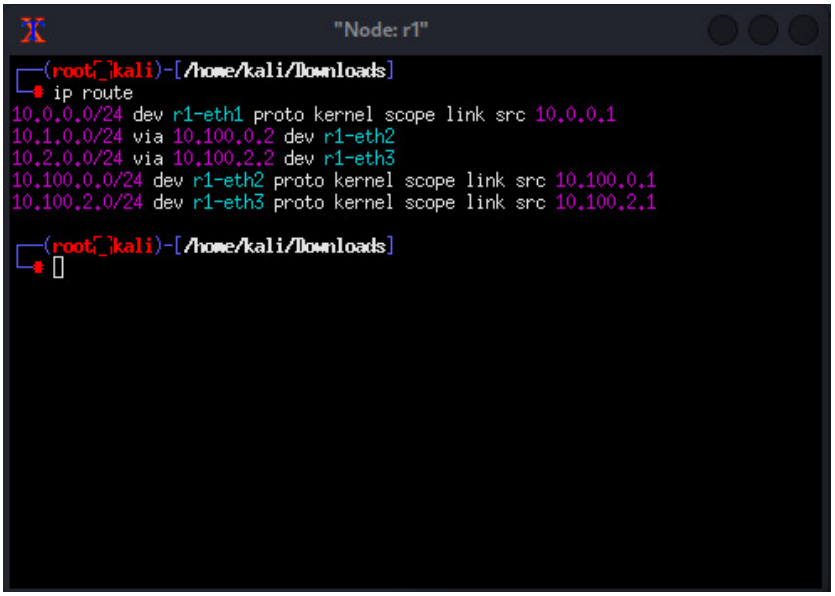

Fig 1.11 : Routing table for router r1 (direct connection)

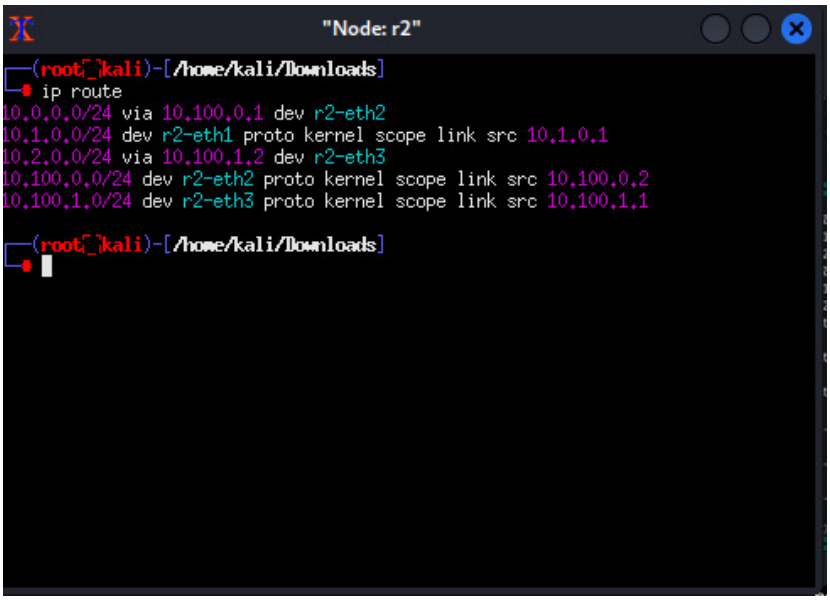

Fig 1.12: Routing tavle for router r2 (direct connection)

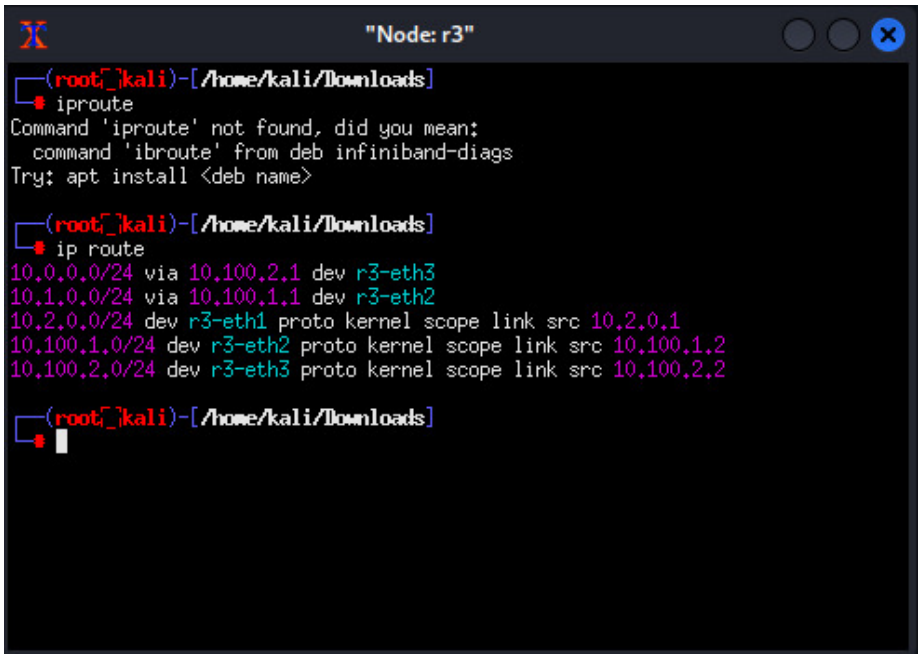

Fig 1.13: Routing table for router r3 (direct)

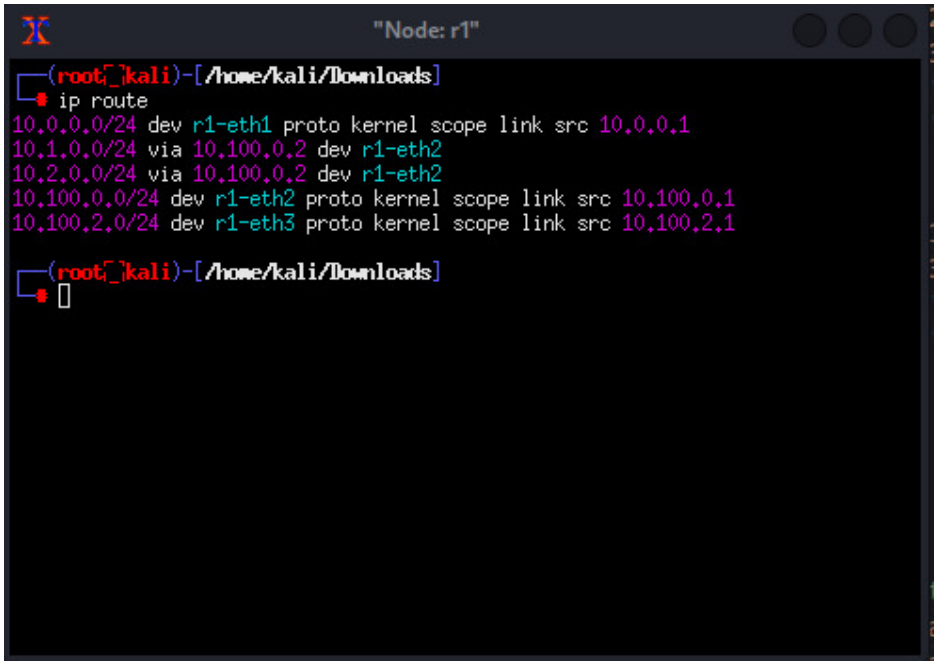

Fig 1.14: Routing table for router r1 (long path)

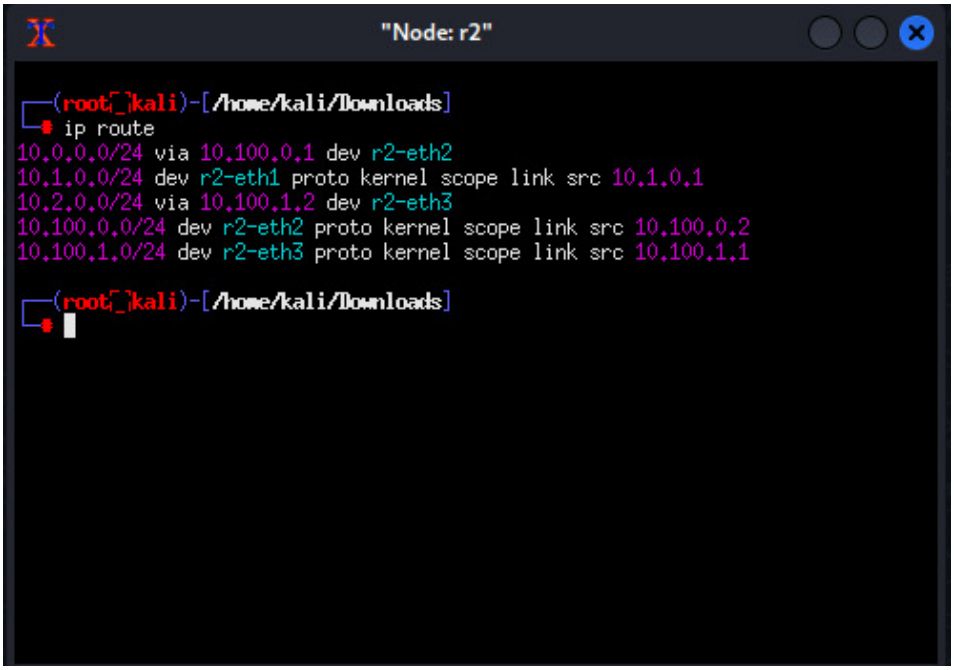

Fig 1.15 : Routing table for router r2 (long path)

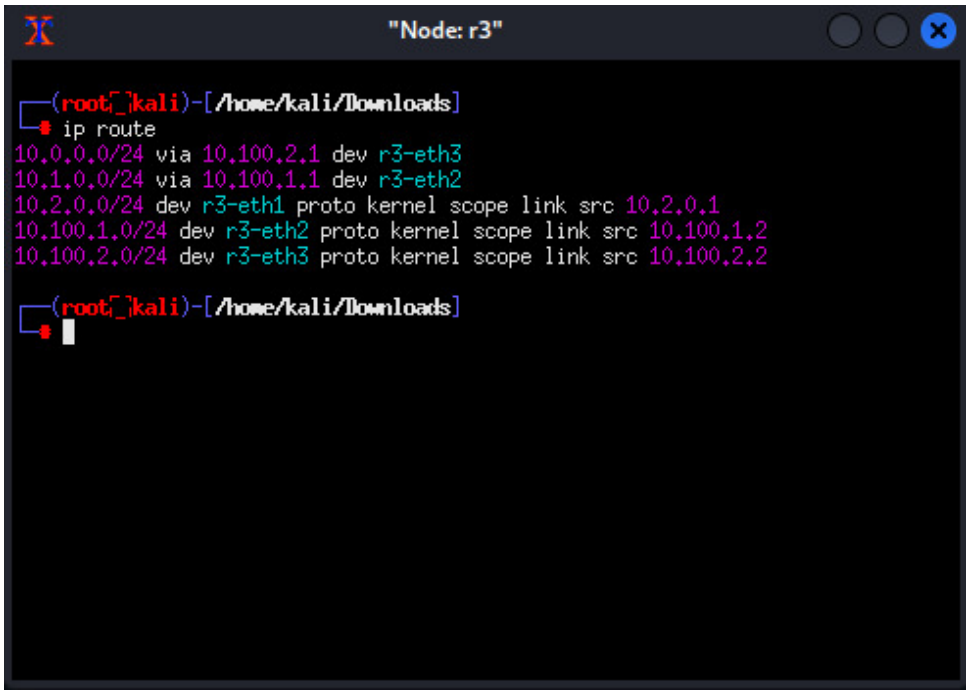

Fig 1.16 : Routing table for router r3 (long path)

In the given figures we can see the routing tables, any packet arriving at r1 with destination r3 will have to go via r2 and then to r3.

Answer 2:

In question 2 we had to implement the topology as shown in the given figure below:

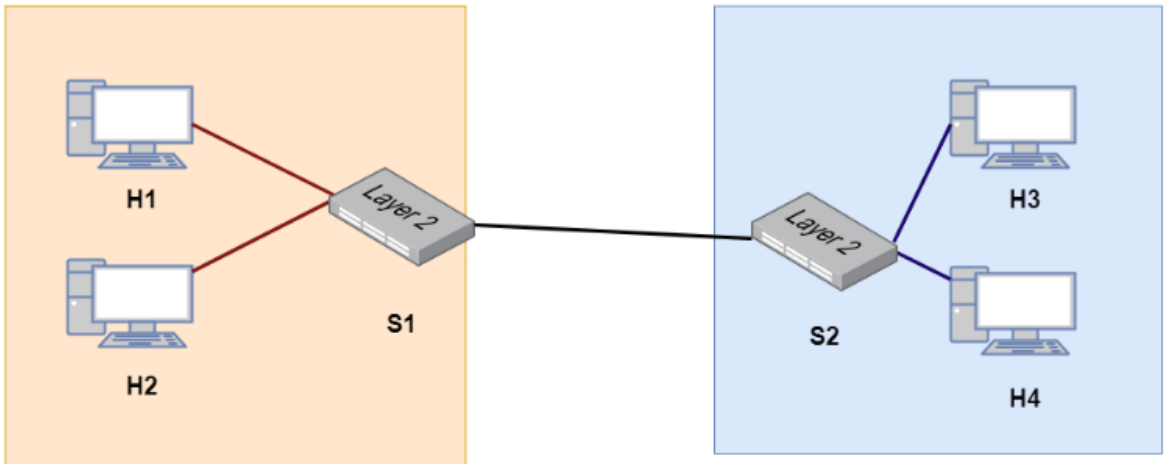

Fig 2.1: Topology for the question

To implement this topology we used mininet Python API, we connected h1,h2 host to a switch s1 and h3,h4 host to switch s2. And we established a link between s1 s2. Now to do the following questions, the tool of iperf was used, iperf can be used to set up a TCP server and clients can connect to the server using iperf. To perform the given questions we used the cmd function to enter the command from within the code. This established a connection between the hosts. The tool iperf has an option to set the type of congestion control algorithm to be used for the connection. We used that setting of iperf to describe the type of congestion control algorithm to be used. To show the details of throughput, we ran the tcpdump command on each of the connecting clients, and we stored the contents in a client.pcap file which then we used in wireshark to plot the figures. We also tried to plot the change in bandwidth plot using the logs given by iperf command but this was not asked in the question so we disabled the function but the code is present and can be enabled by setting the plotall variable to be true. To pass the arguments through command line with the form –config=a we used the library argsparser.

Running the server on h4 and connecting h1, plotting the throughput curve for various congestion algorithms:

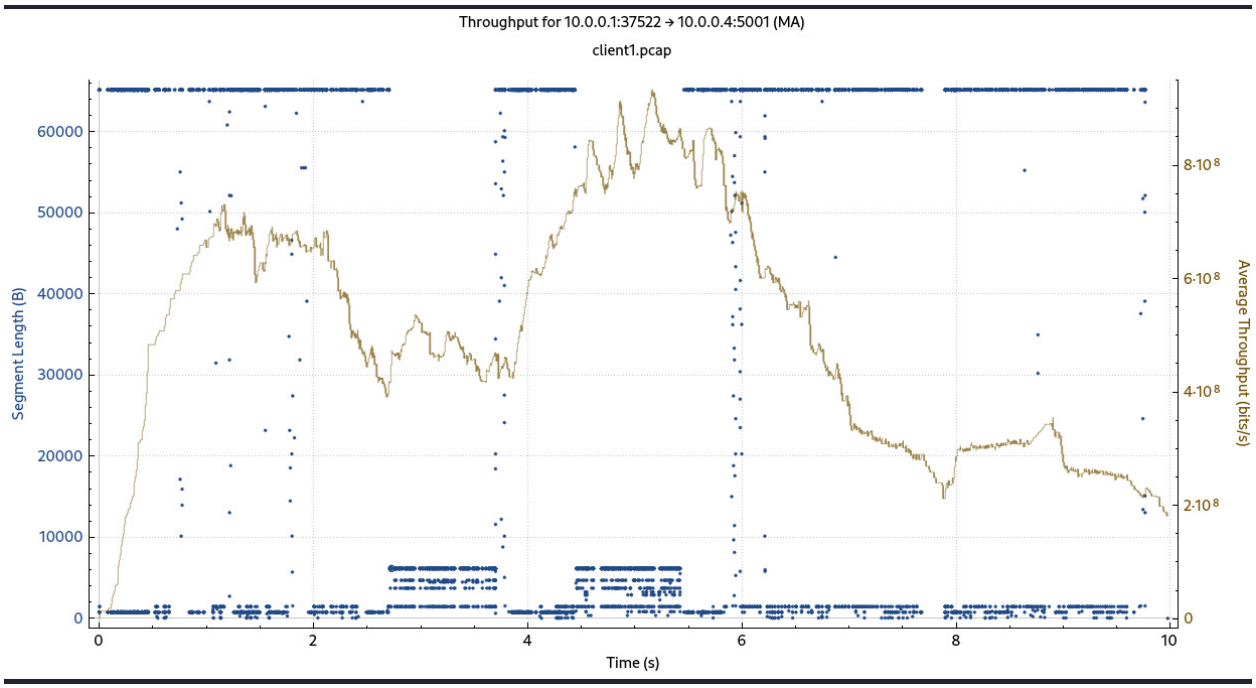

Fig 2.2 : Client throughput using reno

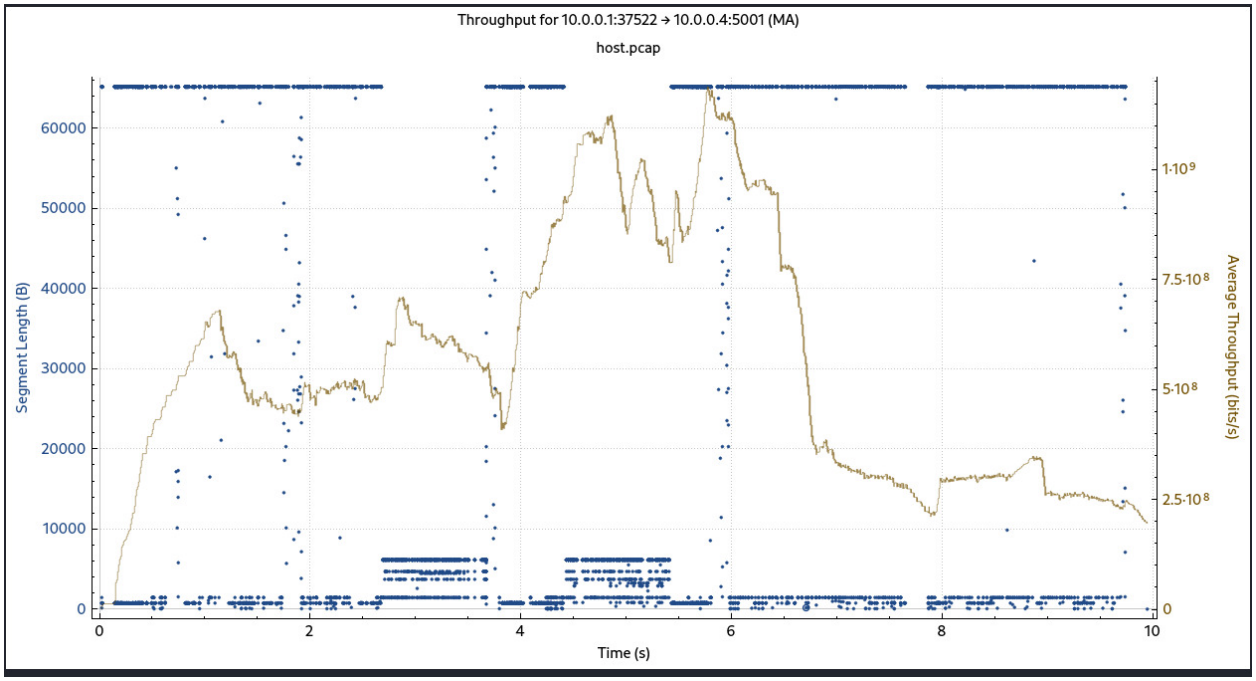

Fig 2.3 : Server throughput using reno

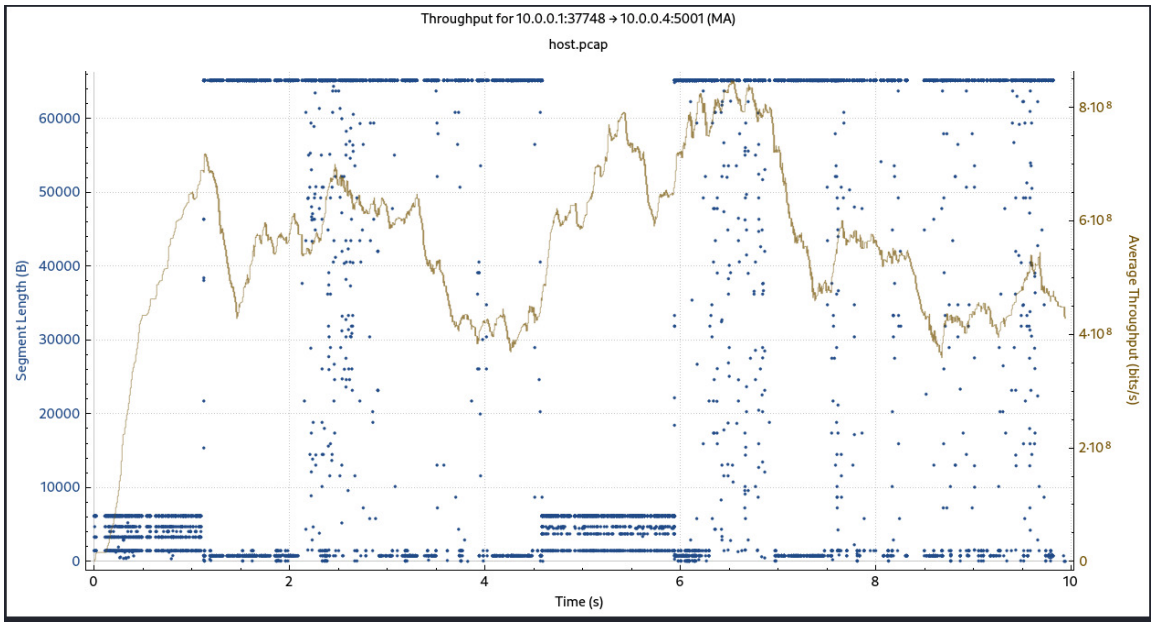

Fig 2.4 : Server throughput using vegas

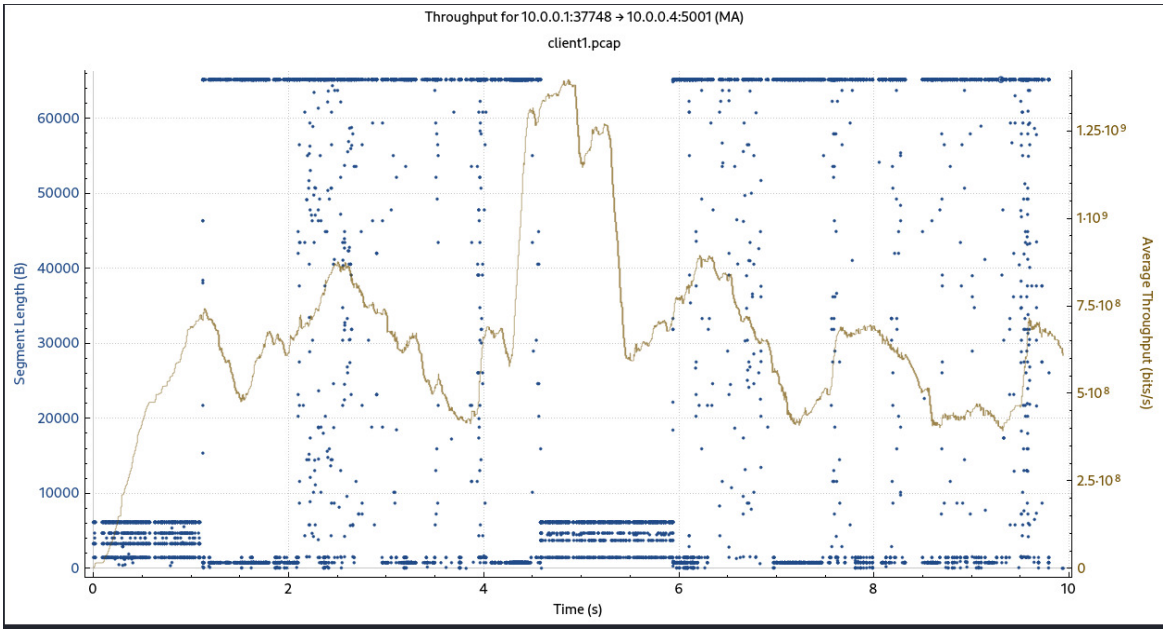

Fig 2.5: Client throughput using vegas

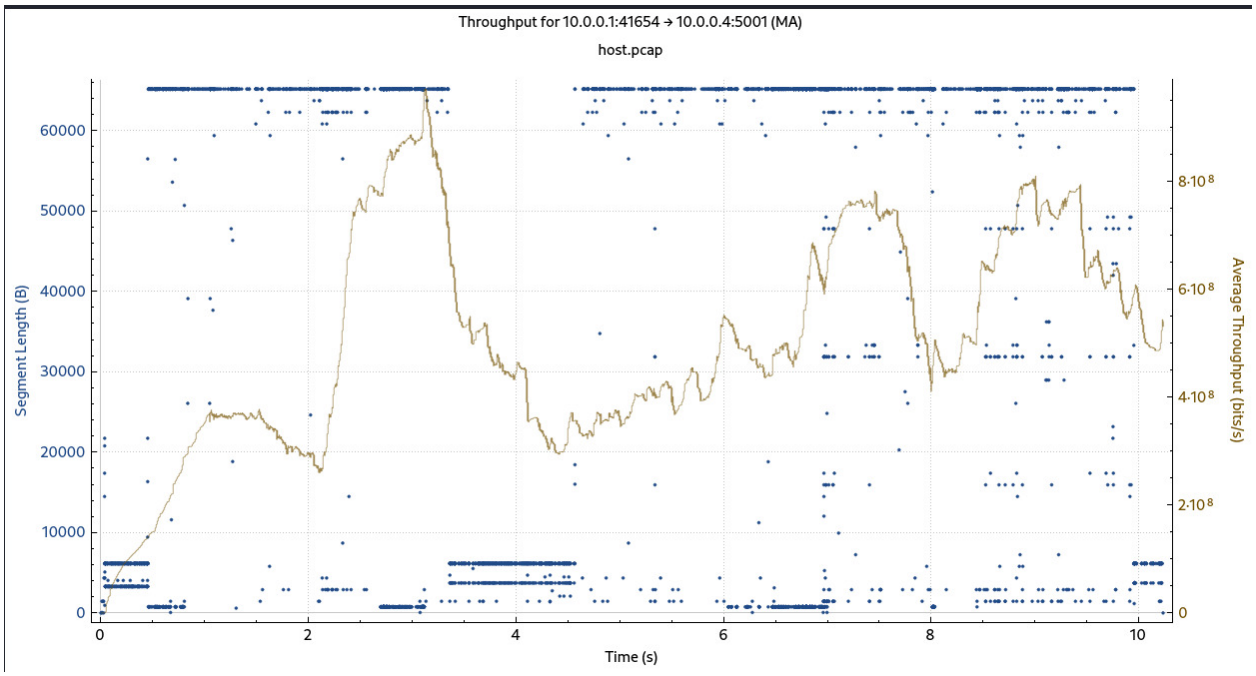

Fig 2.6: Host throughput using bbr

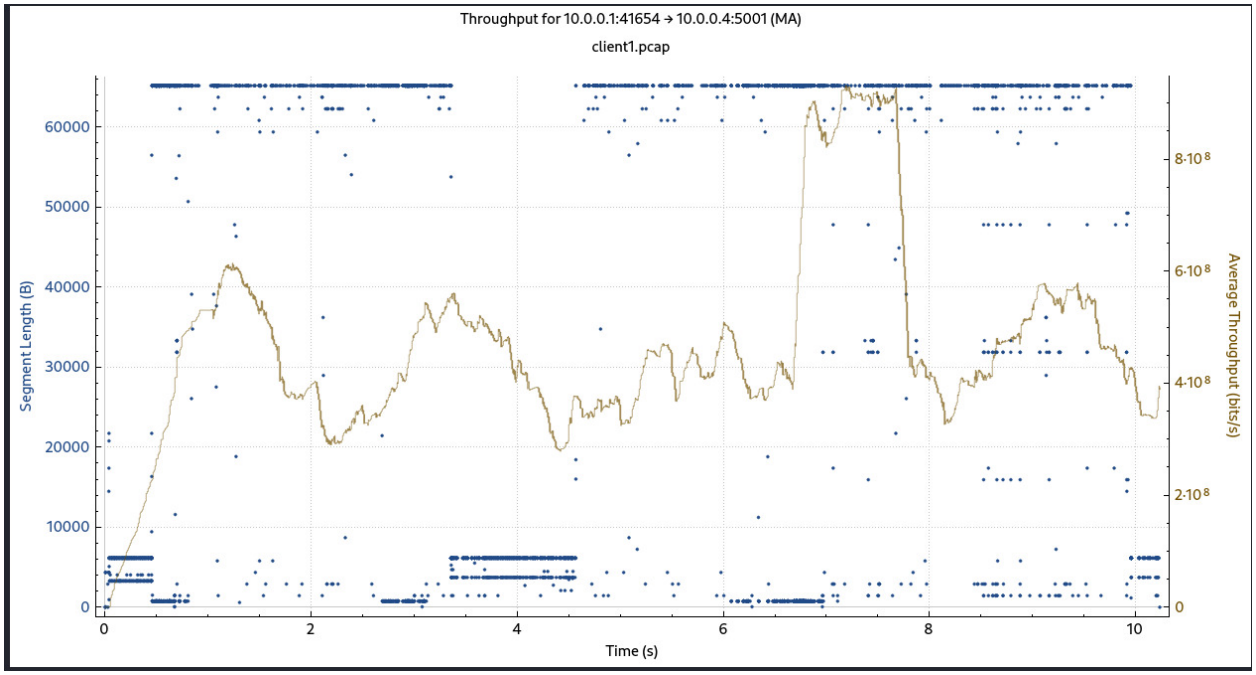

Fig 2.7: Client throughput using bbr

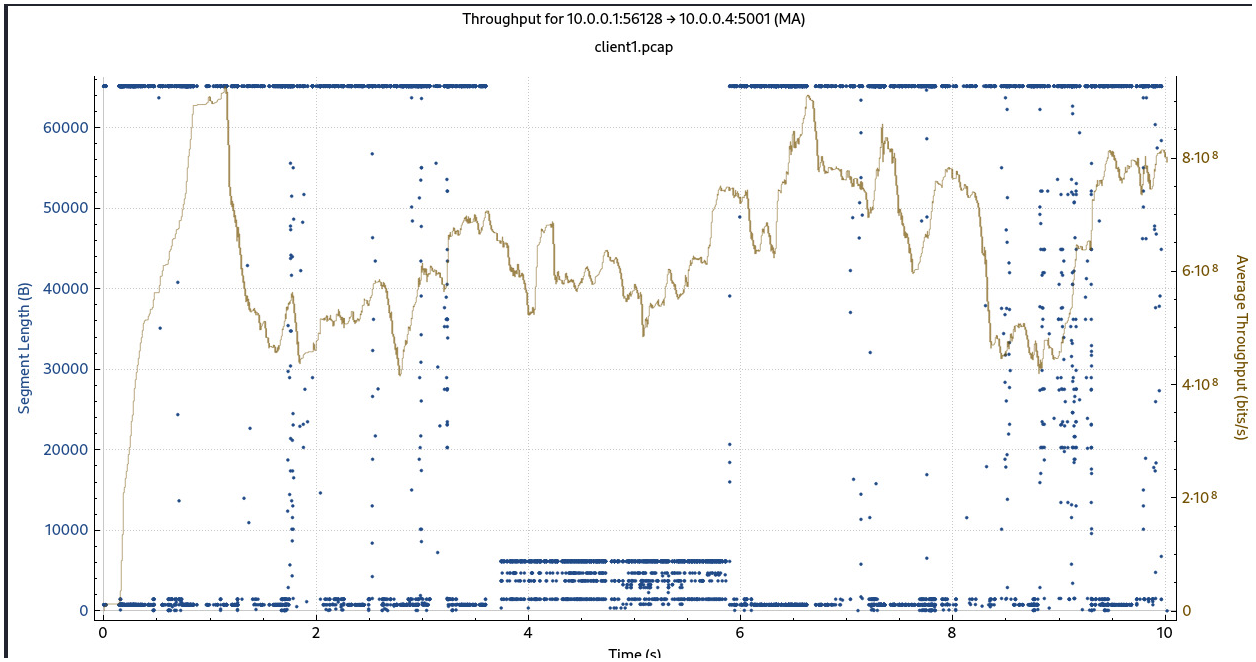

Fig 2.8: Client throughput using cubic

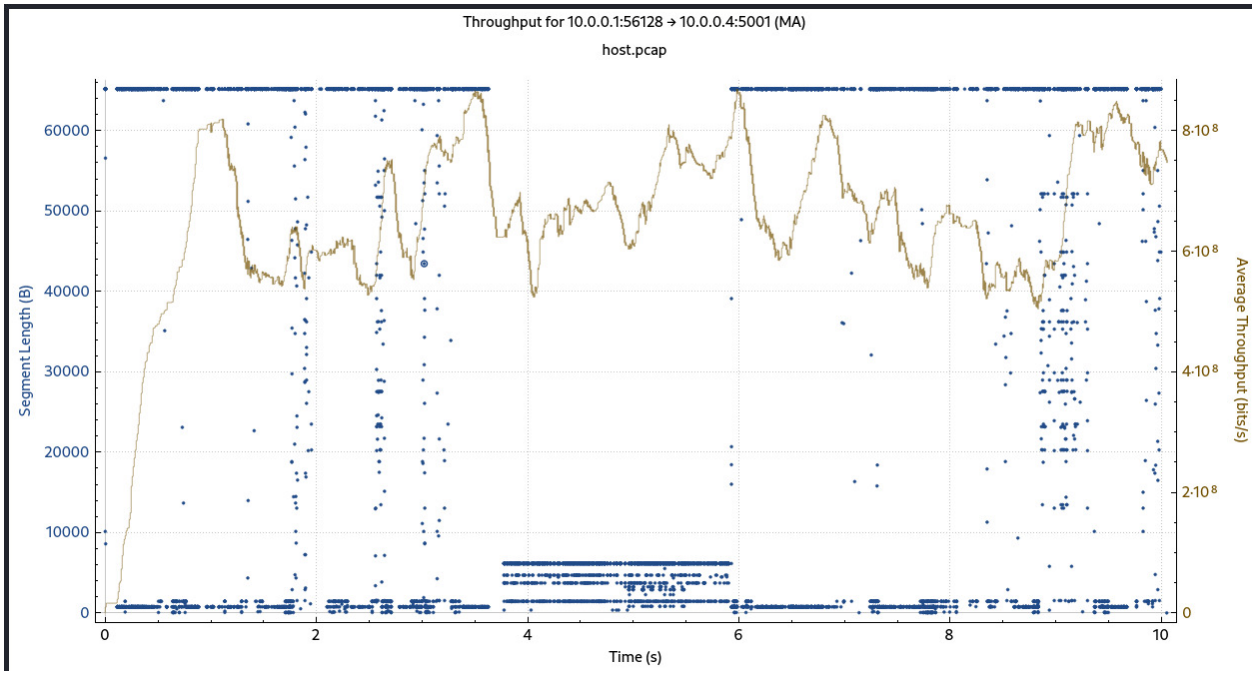

Fig 2.9: Server throughput using cubic

Reasoning:

In the case of reno from the plot we can clearly see that during the initial phase the segment length increases and the throughput increases because of slow start phase in which the congestion window increases exponentially, and after a point we can see the congestion avoidance phase where the increase is approximately lineary increasing.

After the congestion we can see the decrease in the throughput which is expected, and then the congestion window is increased.

In the case of vegas we can see the segment size is maintained between a range and we can also see that the complete bandwidth utilization is not being done when compared to tcp reno, this is because the vegas algorithm uses the rtt to increase or decrease the congestion window it tries to prevent packet loss and hence we can see lower throughput and fewer drops.

Similarly for the plot of bba we can see the results of the buffer based approach where the entry of the packets is controlled into the network, it results in fair allocation of bandwidth but in single host connection we cant see much of difference.

The cubic algorithm differs from the reno algorithm in congestion avoidance phase, wherein in the cubic algorithm the congestion window is increased in the cubic power if the difference between the current window and the window where congestion occurred, due to which we can observe that the increase in throughput after packet loss is more than in other algorithms because it is cubic.

Answer C part simultaneously run

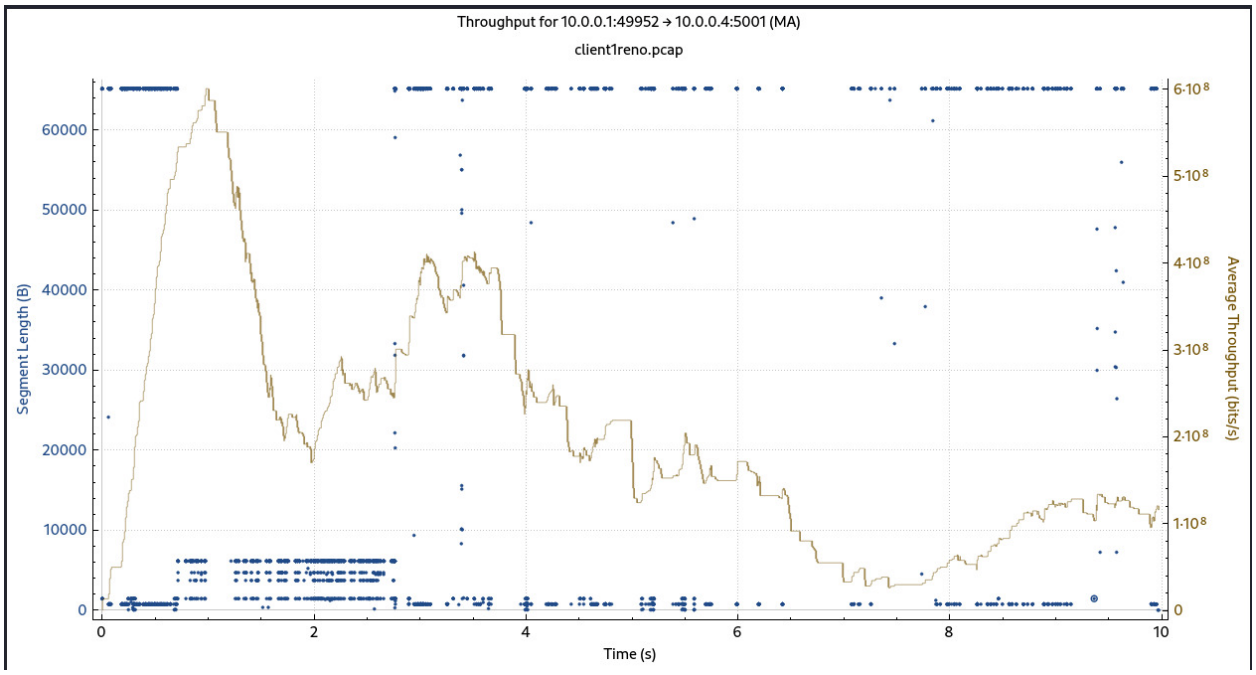

Fig 2.10: Client 1 throughput using reno

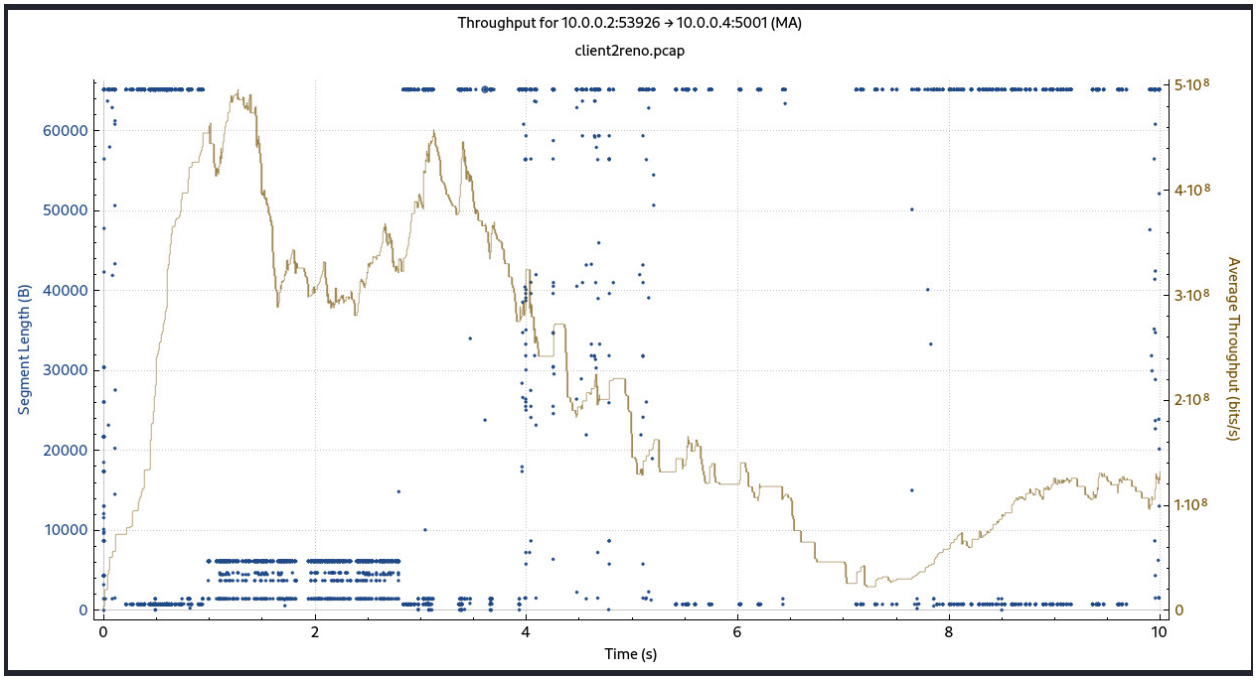

Fig 2.11 Client 2 throughput using reno

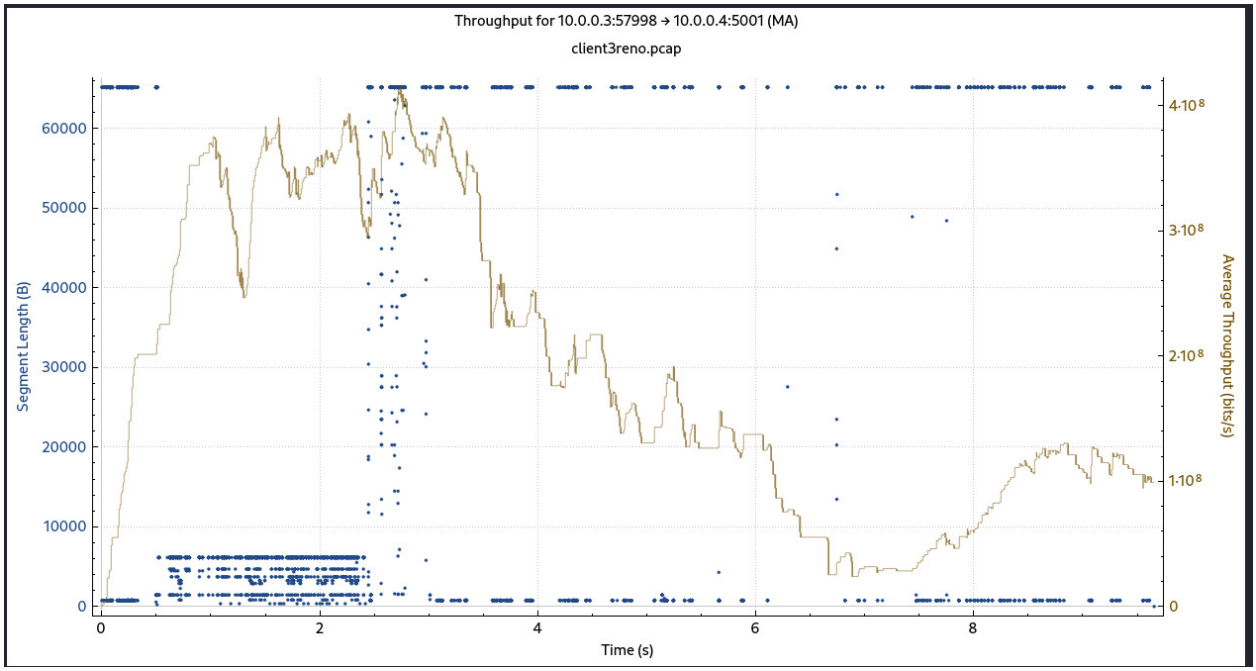

Fig 2.12 Client 3 throughput using reno

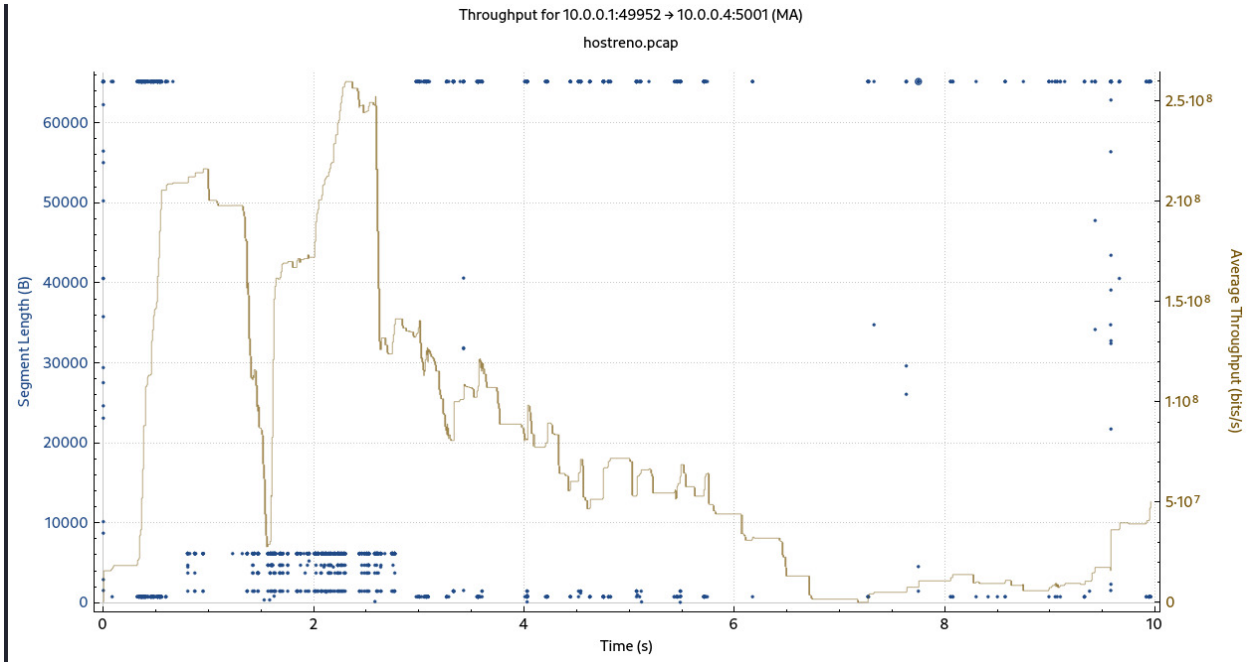

Fig 2.13 Server throughput on one of the stream using reno

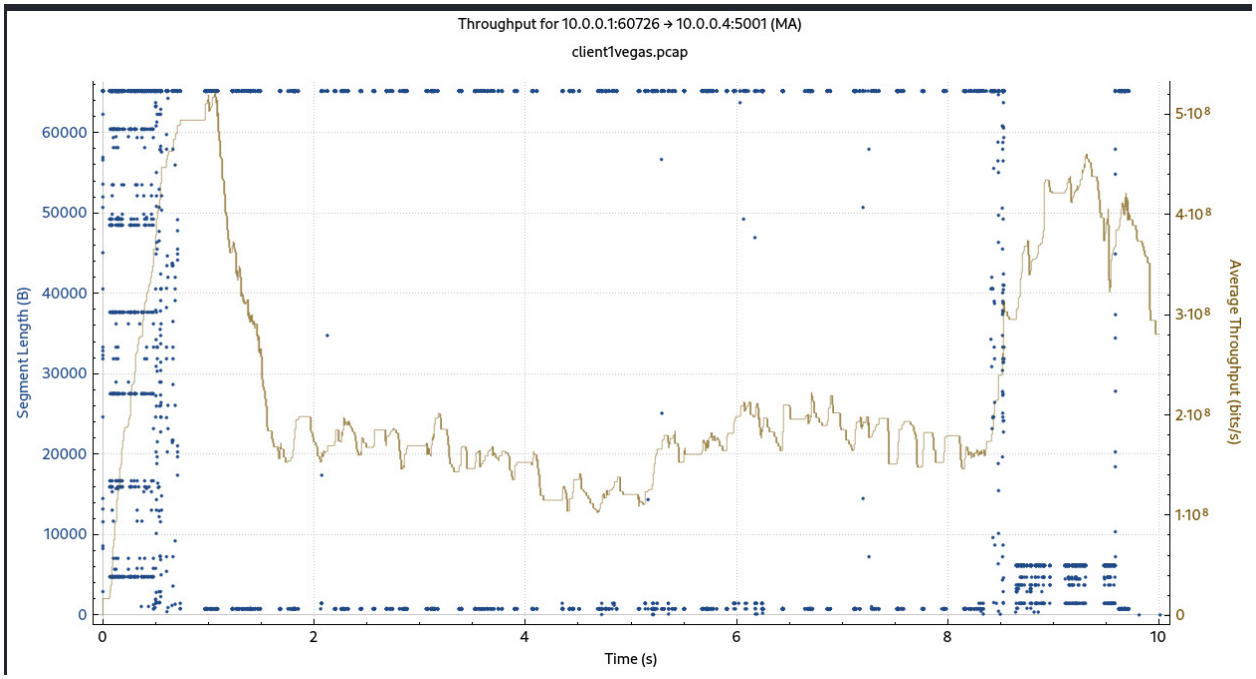

Fig 2.14 Client 1 throughput using vegas

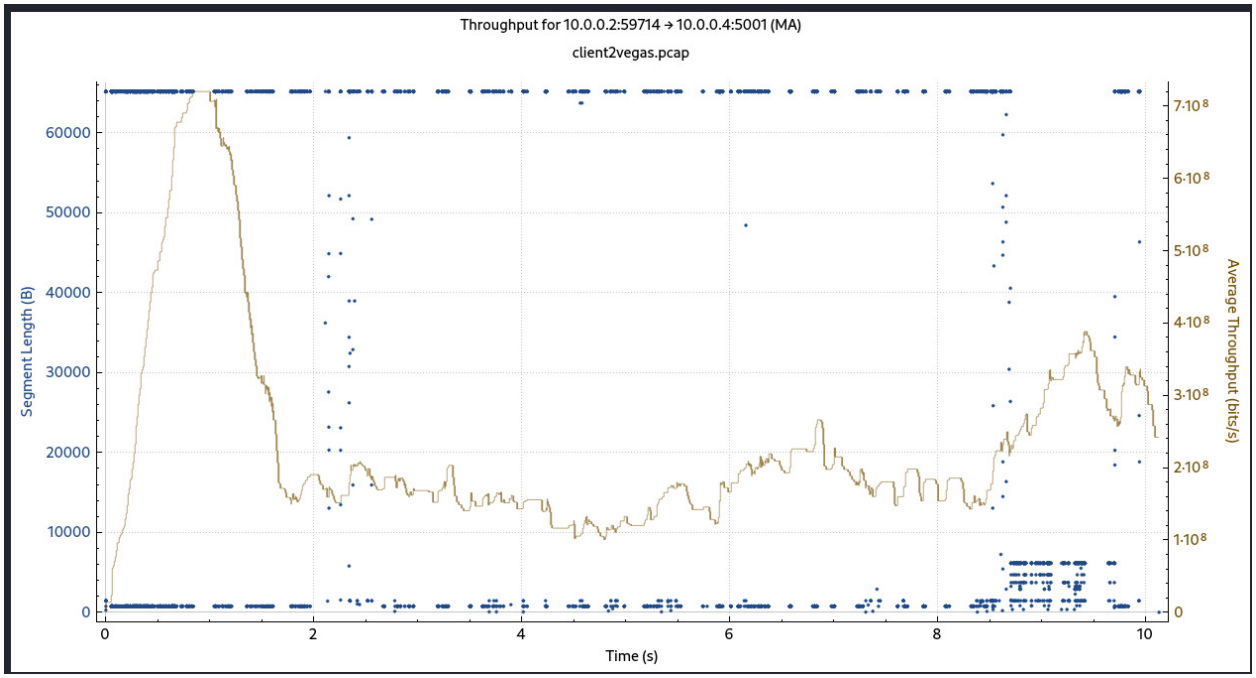

Fig 2.15 Client 2 throughput using vegas

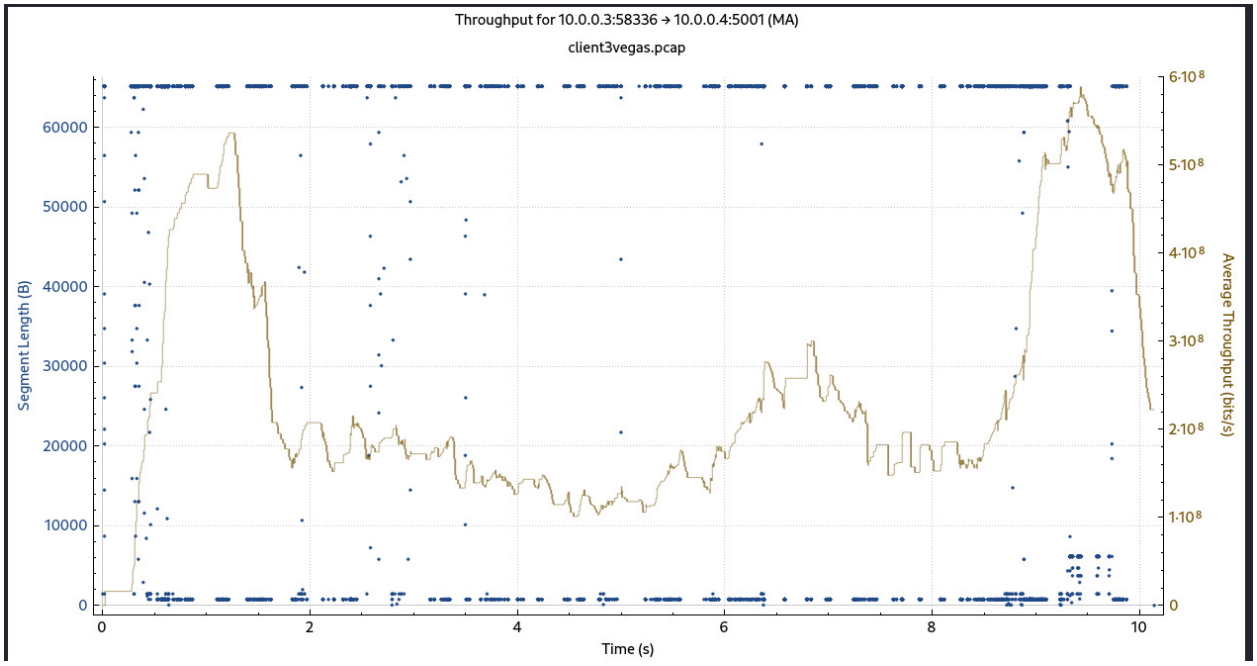

Fig 2.16: Client throughput using vegas

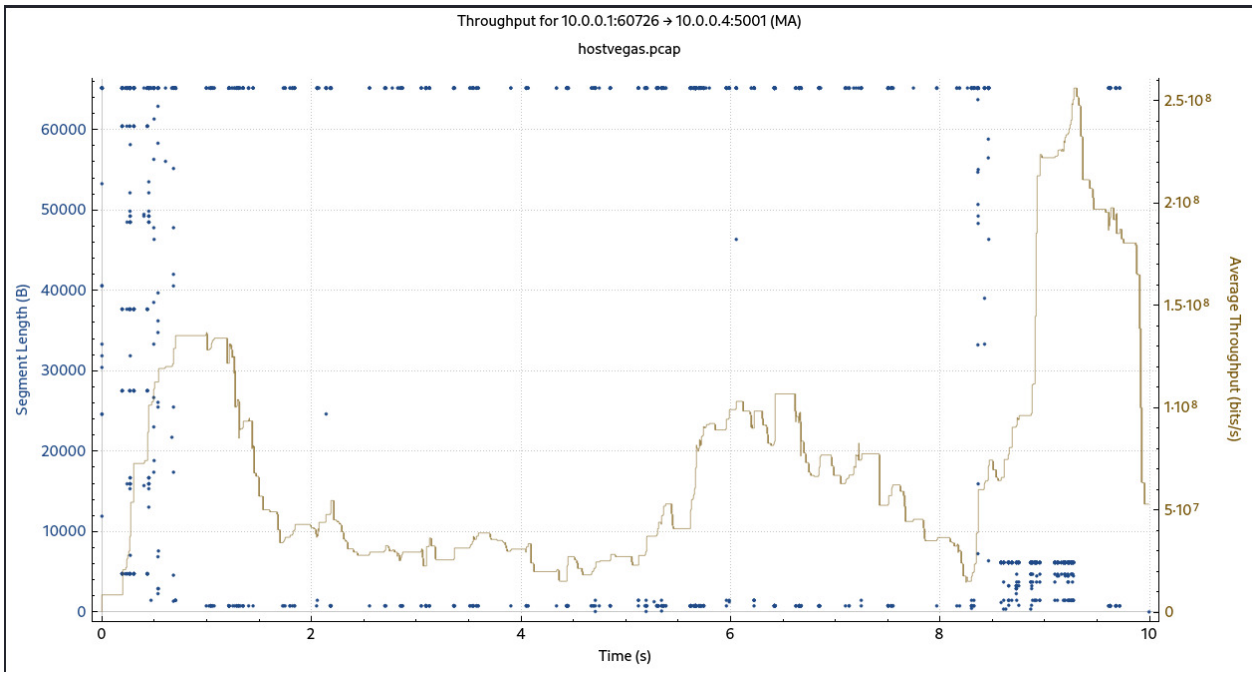

Fig 2.17: Server throughput using vegas

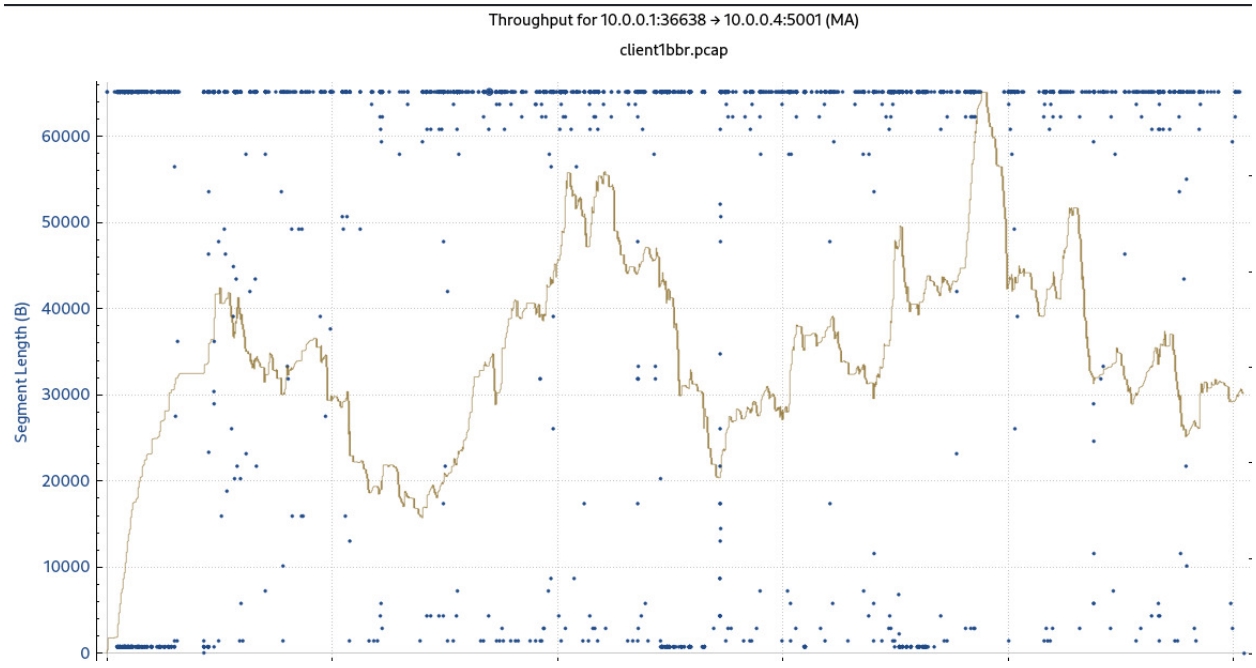

Fig 2.18: Client 1 throughput using bbr

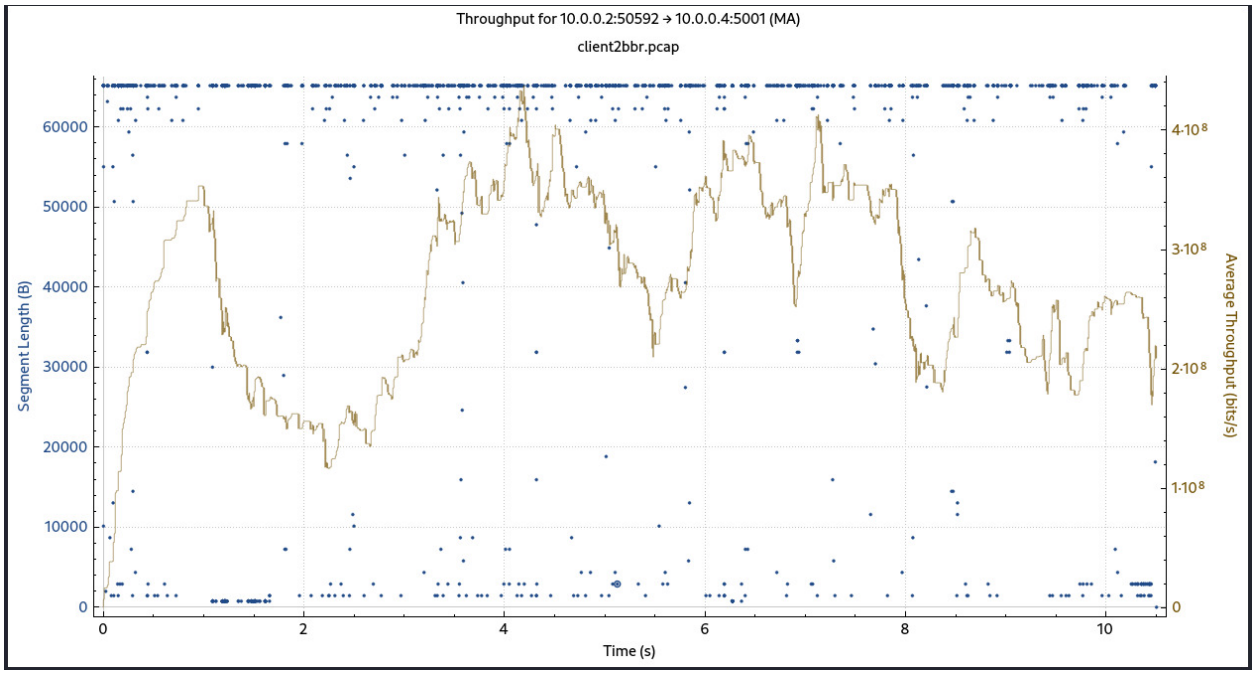

Fig 2.19 Client 2 throughput using bbr

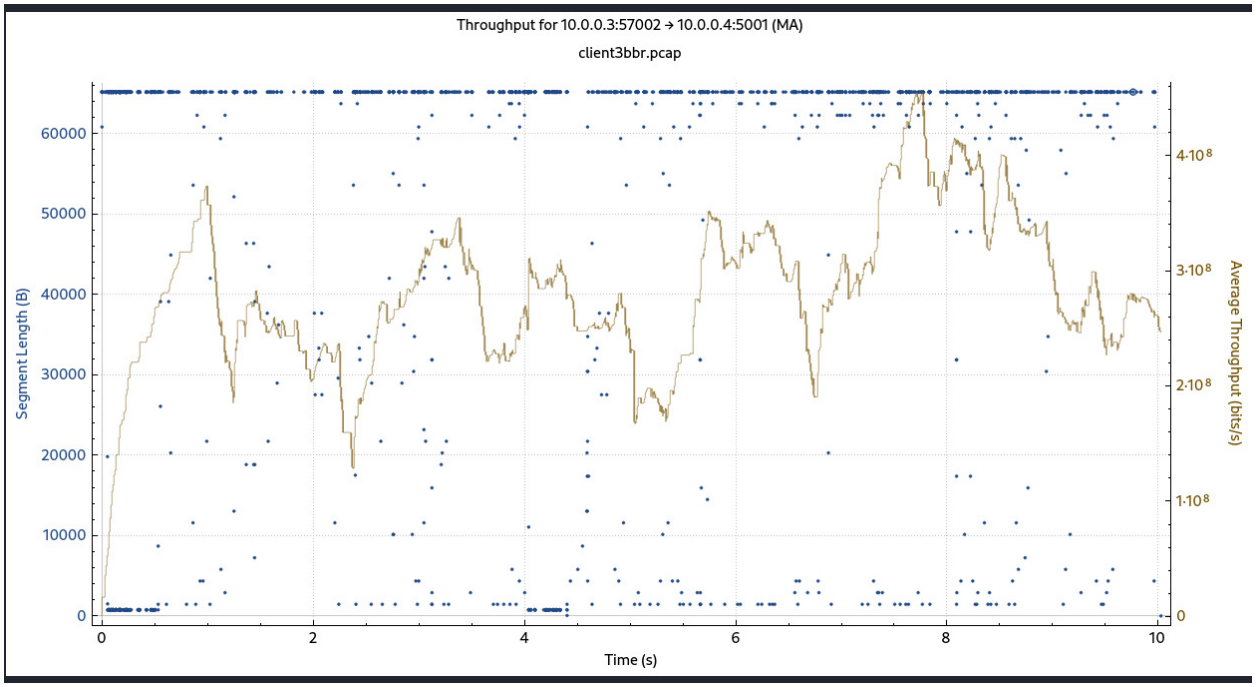

Fig 2.20: Client 3 throughput using bbr

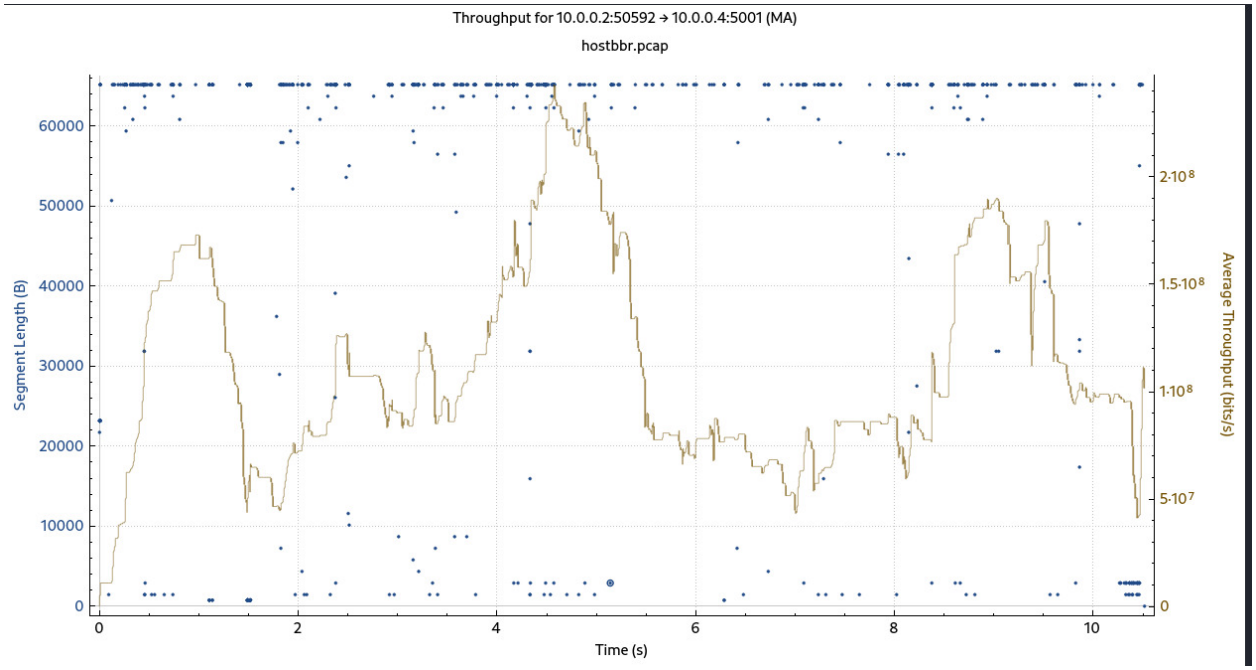

Fig 2.21 Server throughput using bbr on a single stream

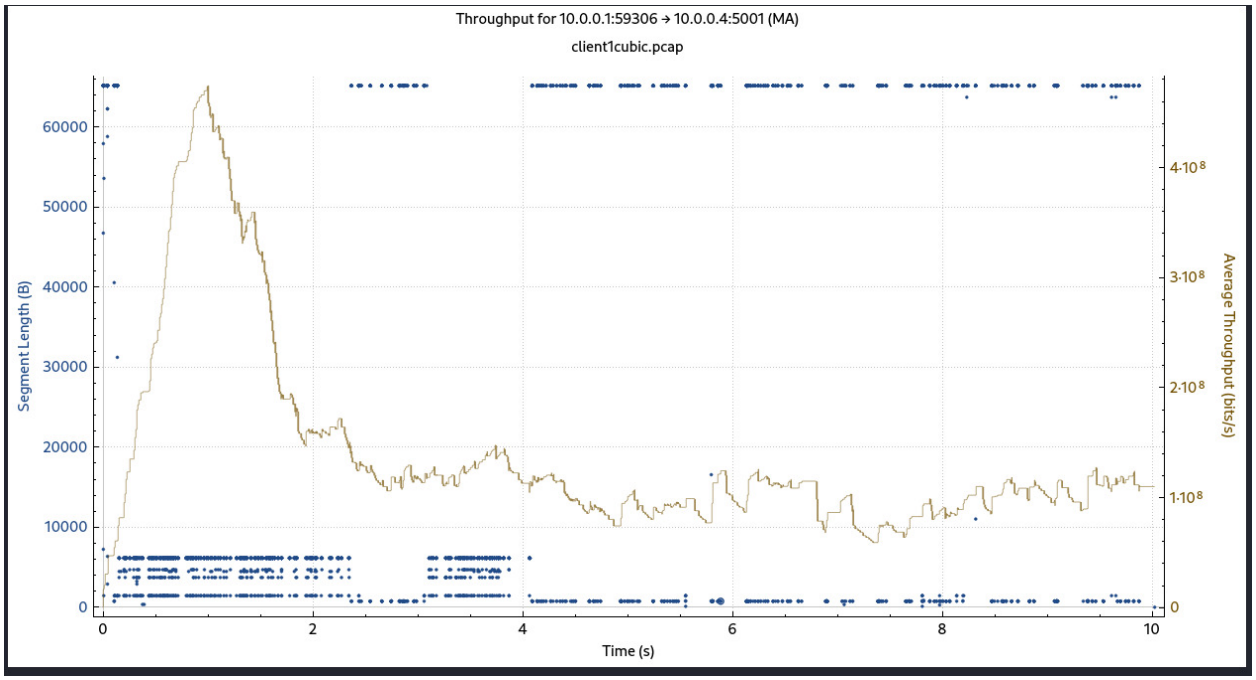

Fig 2.22 Client 1 throughput using cubic

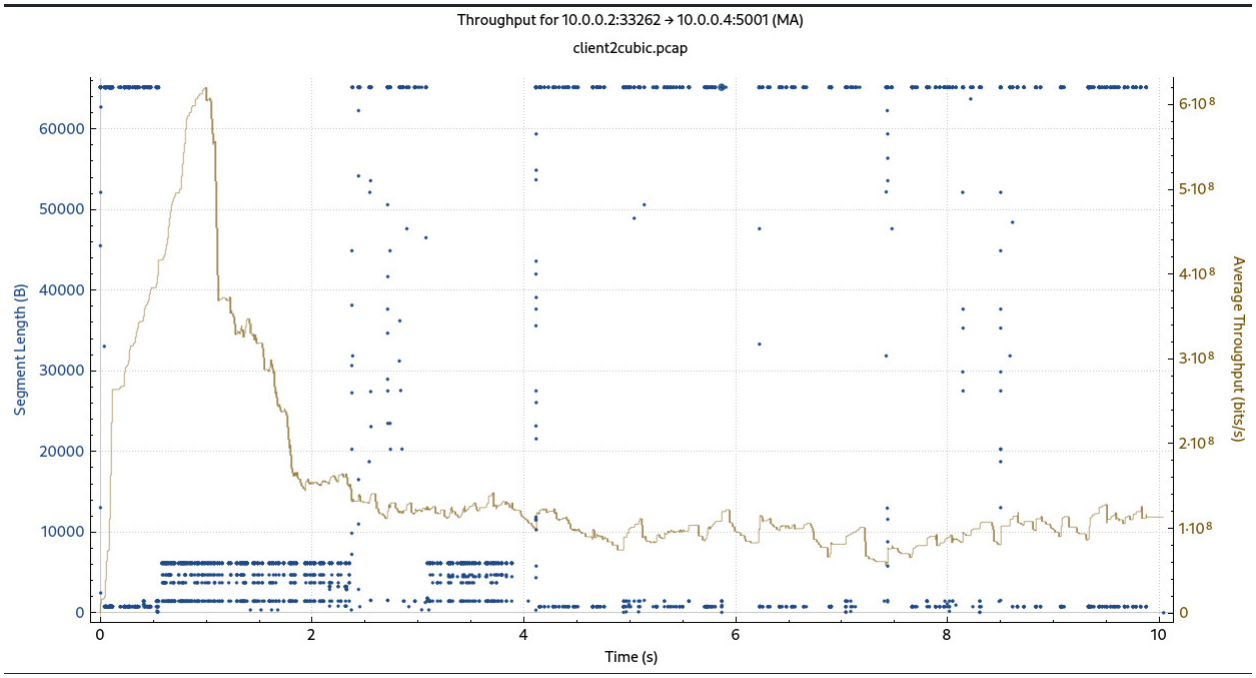

Fig 2.23 Client 2 throughput using cubic

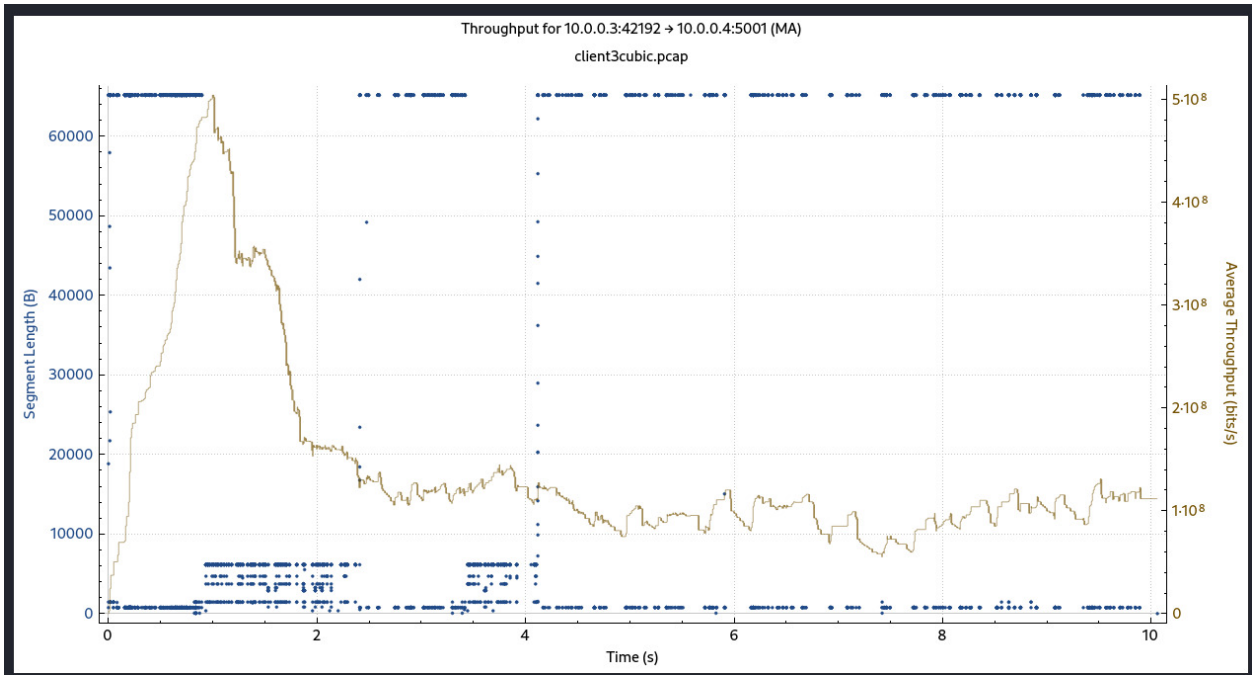

Fig 2.24 Client 4 throughput using cubic

Fig 2.25 Server throughput using cubic on a single TCP stream

Reasoning: The above plots can be understood in the following way, in the reno plot we can see that during the slow start phase the segment size is increasing for all the hosts, but as soon as the link bandwidth is reached the congestion window for the clients is reduced and then after the congestion event the window size is constantly decreasing, because as there are 3 clients they compete with each other and congestion event keeps on happening.

In the vegas algorithm we can see that the congestion window is between a particular range, as it is a delay based algorithm when the delay increases beyond a threshold the window size is reduced, also we can see that the increase in window size is not much because it is increased by 1 at every iteration. The spike for 2 clients can be explained in a way that some client disconnected or experienced loss due to which other clients get more bandwidth.

BBR is a delay based algorithm it uses both the observed bandwidth and delay values to adjust the congestion window. Sometimes bbr tries to send more number of packets than the estimated bandwidth. Hence the plot can be explained on similar context.

In the cubic algorithm the increase in congestion window is very fast because of slow start phase, and after congestion occurred the increase in the segment length is cubic, but from the plot we can see that the segment length is constantly decreasing because three clients are constantly running in parallel, and thus each of them tries to increase

the segment size in a cubic manner but they experience congestion due to which the graph is observed. We can also observe that the bandwidth per client is reduced compared to only having a single connection.

Answer d part with link loss

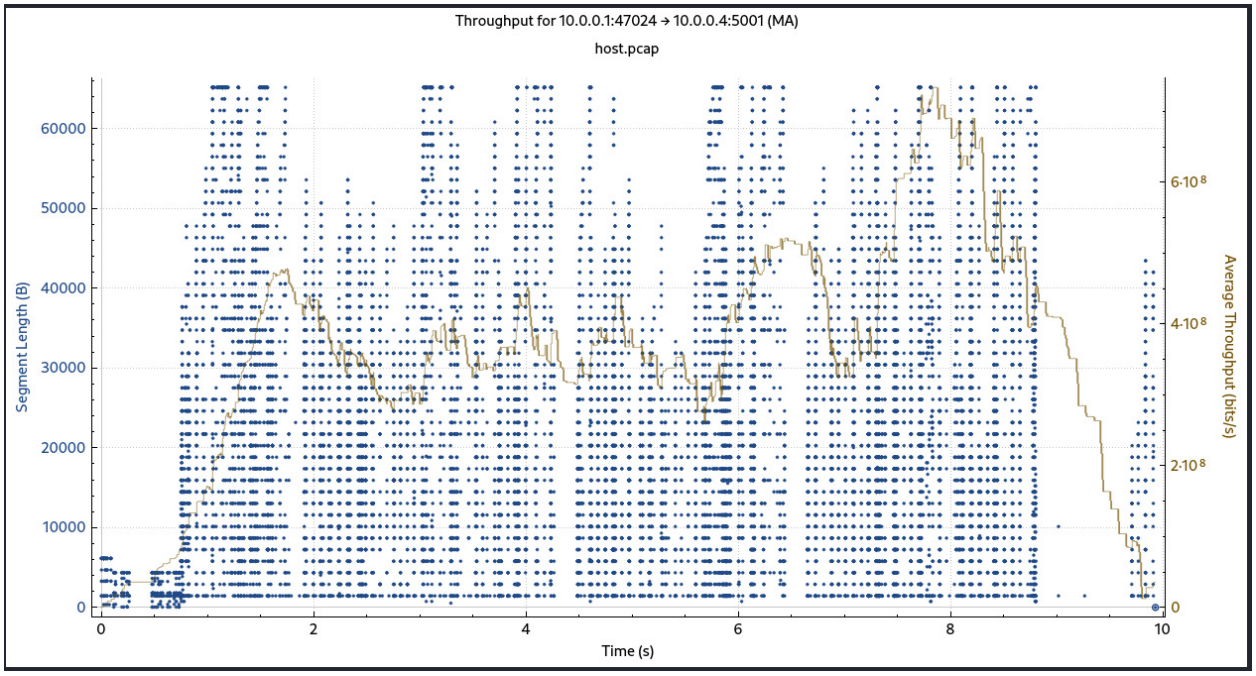

Fig 2.26 : Server throughput using reno with 1% link loss

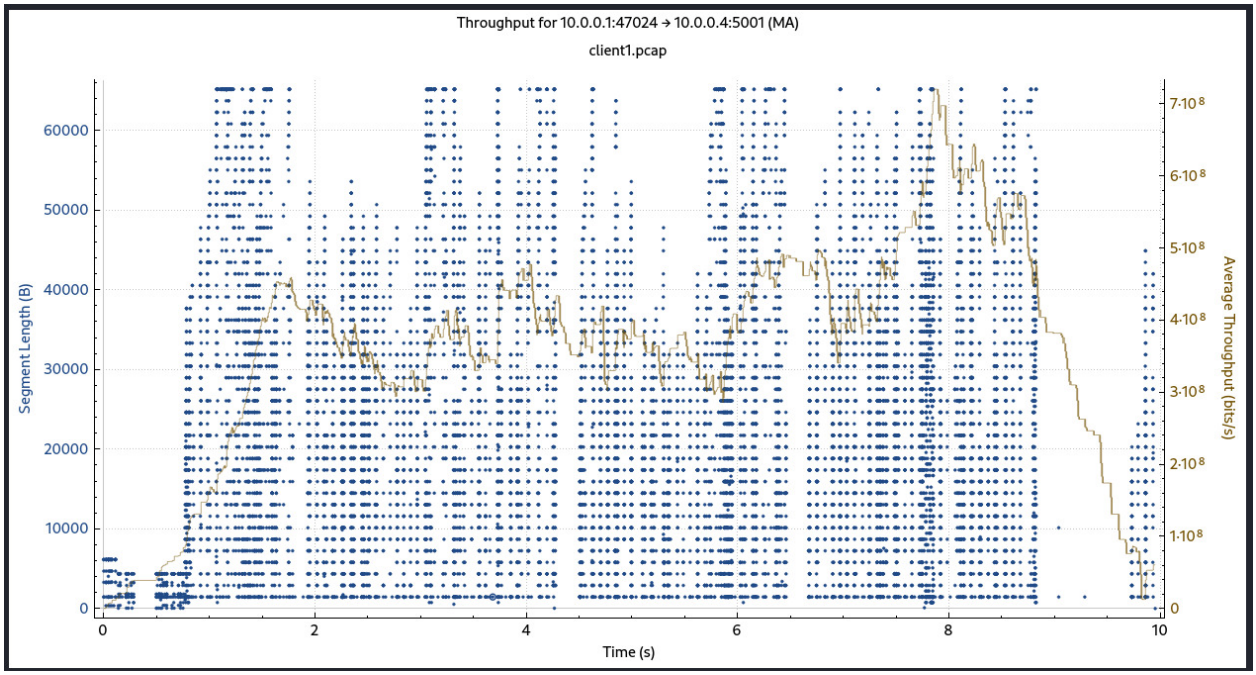

Fig 2.27 Client throughput using reno with 1%link loss

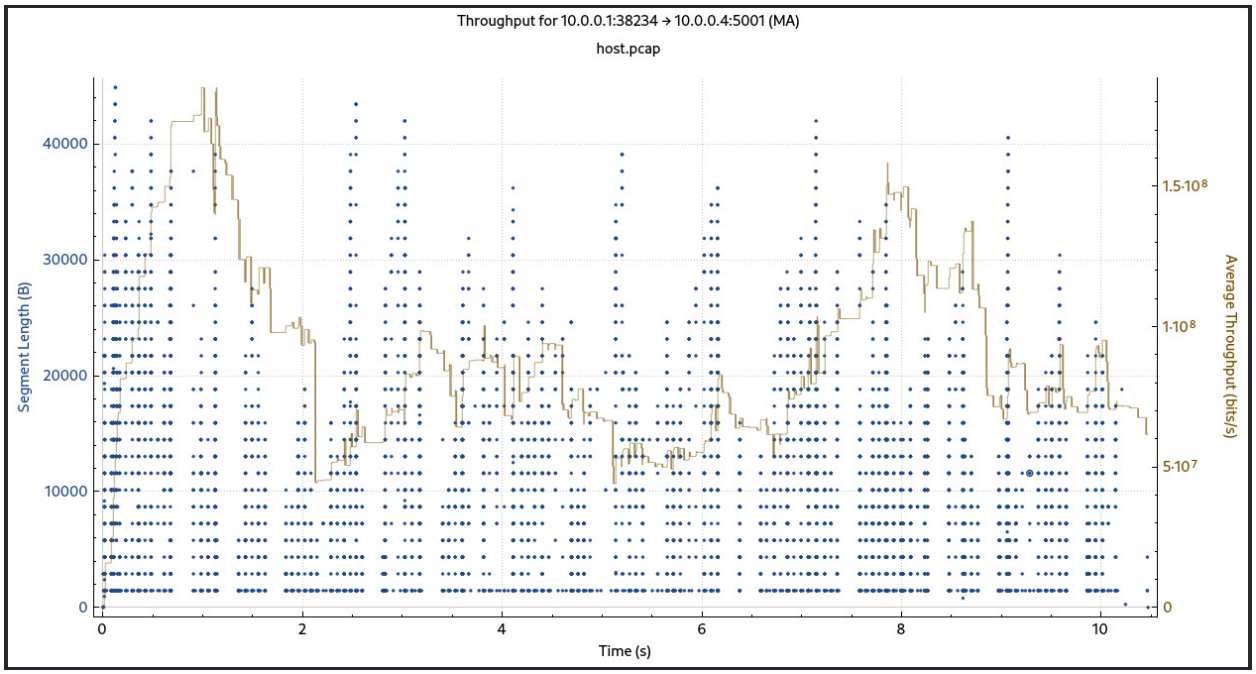

Fig 2.28: Server throughput using reno with 3% link loss

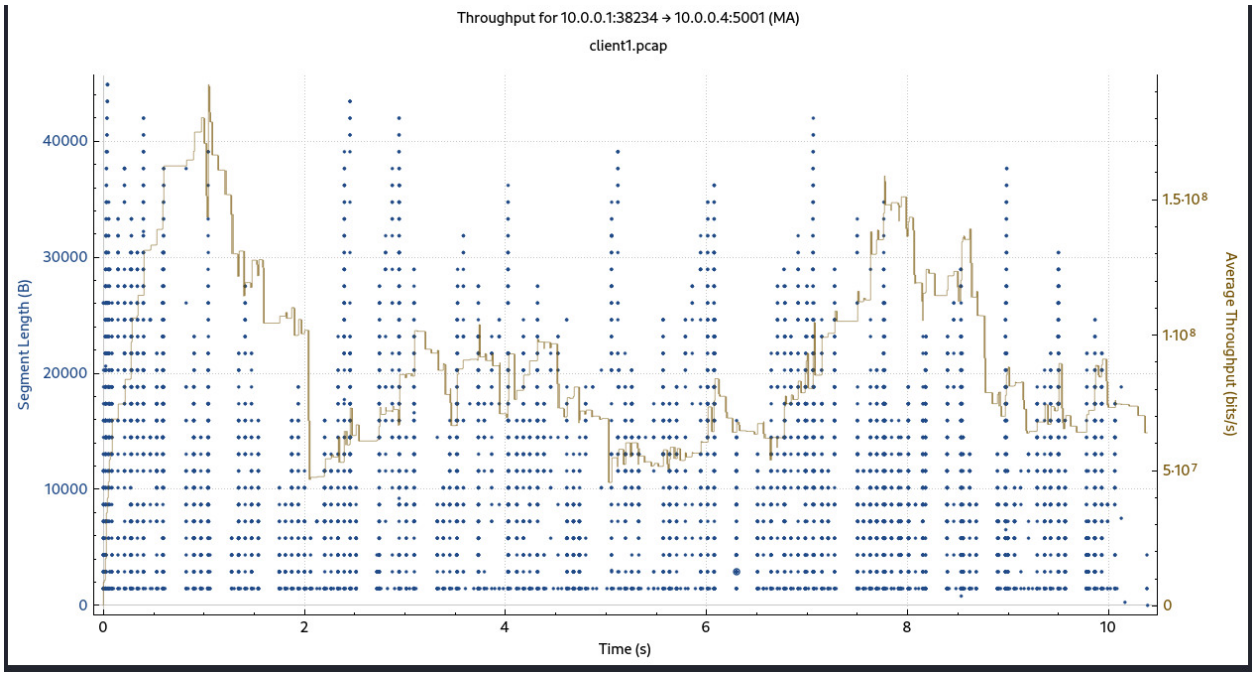

Fig 2.29: Client throughput using reno with 3% link loss

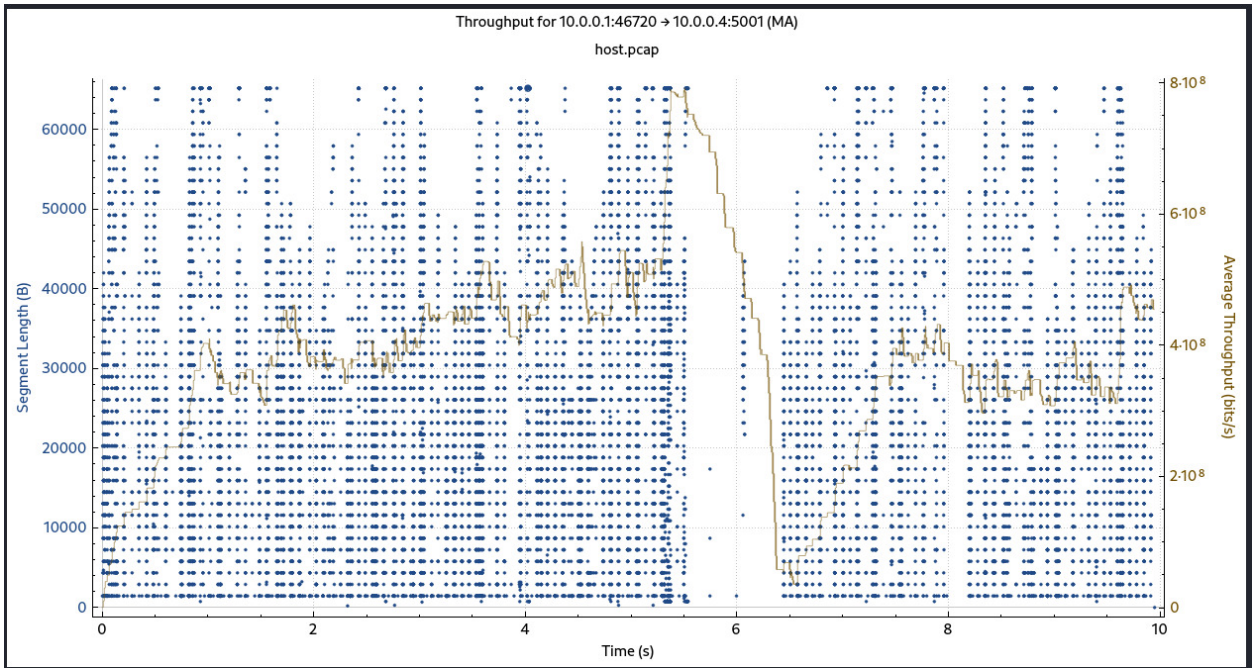

Fig 2.30: Server throughput using vegas with 1% link loss

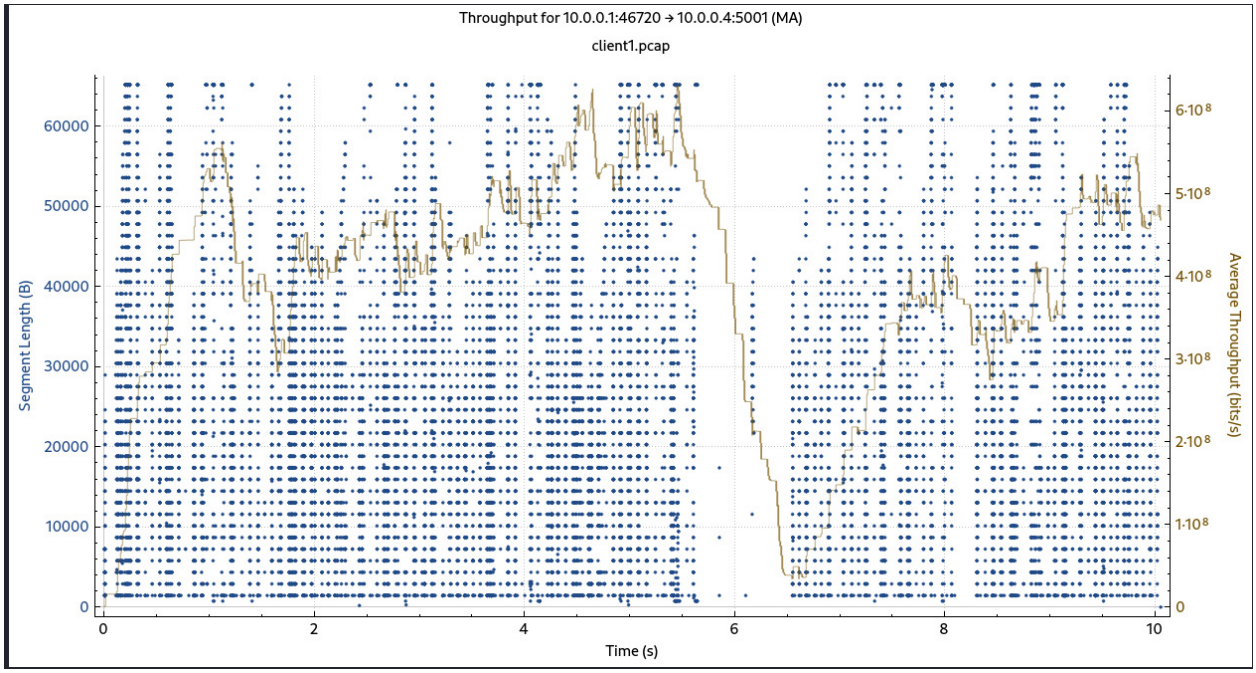

Fig 2.31 Client throughput using vegas with 1% link loss

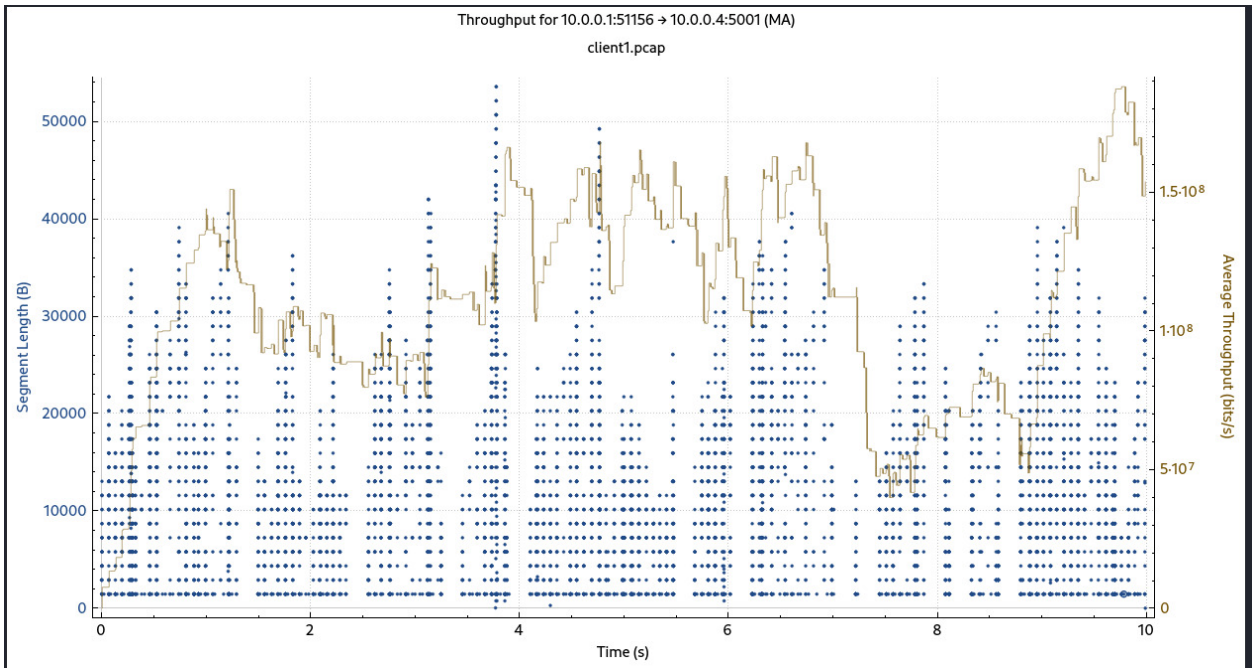

Fig 2.32 Client throughput using vegas with 3% link loss

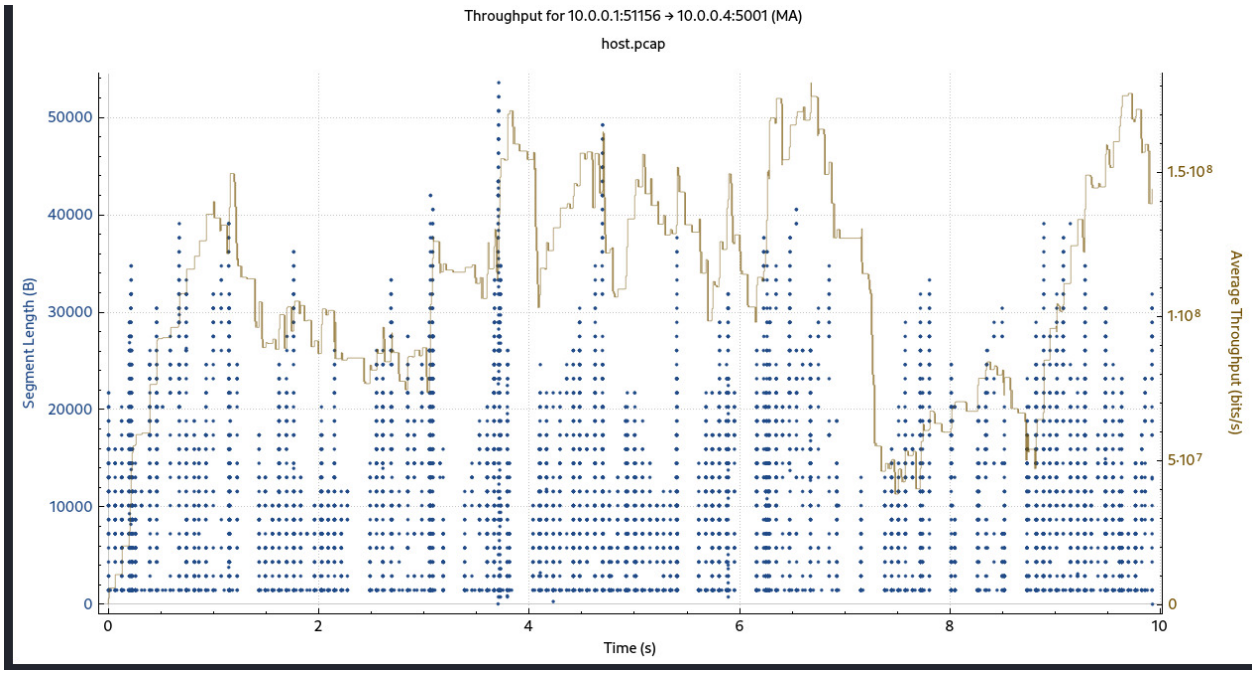

Fig 2.33 Server throughput using vegas with 3% link loss

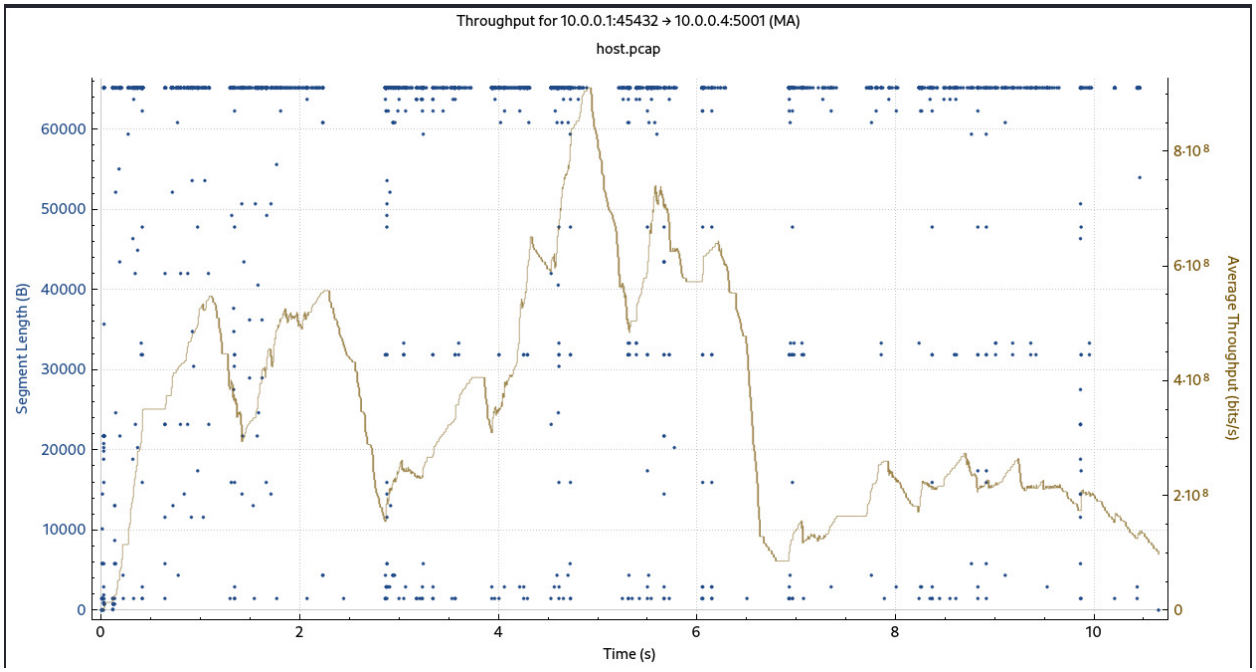

Fig 2.34: Server throughput using bbr with 1% link loss

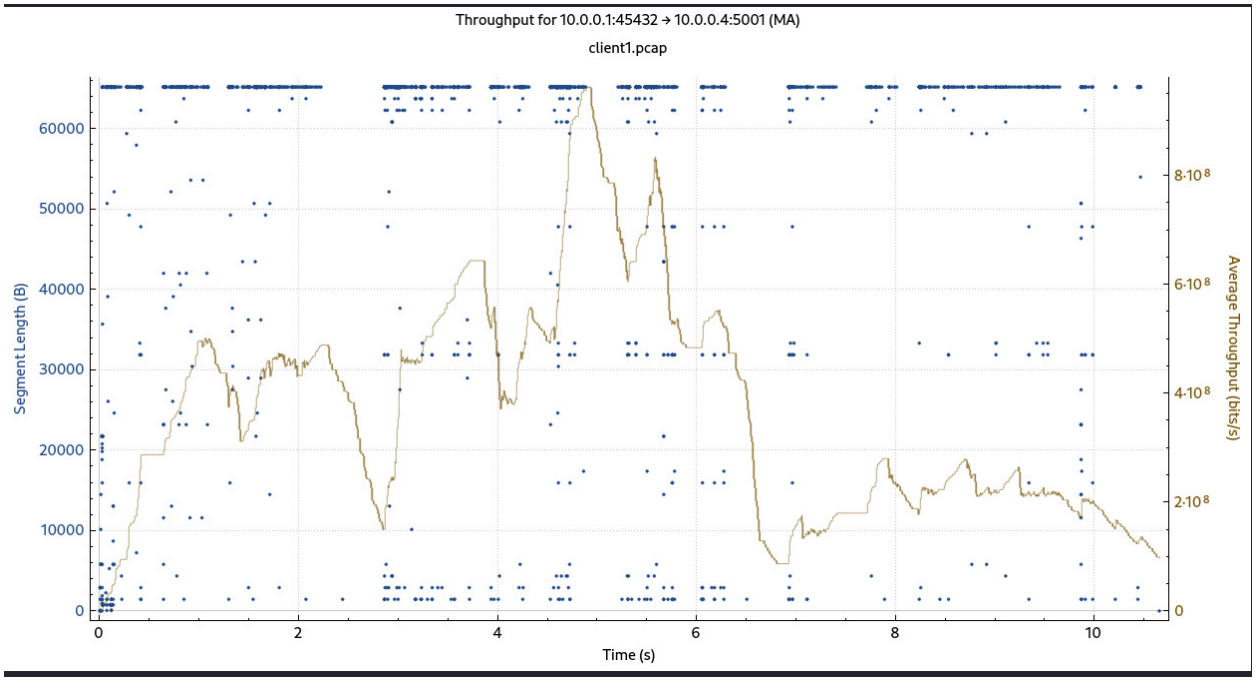

Fig 2.35 Client throughput using bbr with 1% link loss

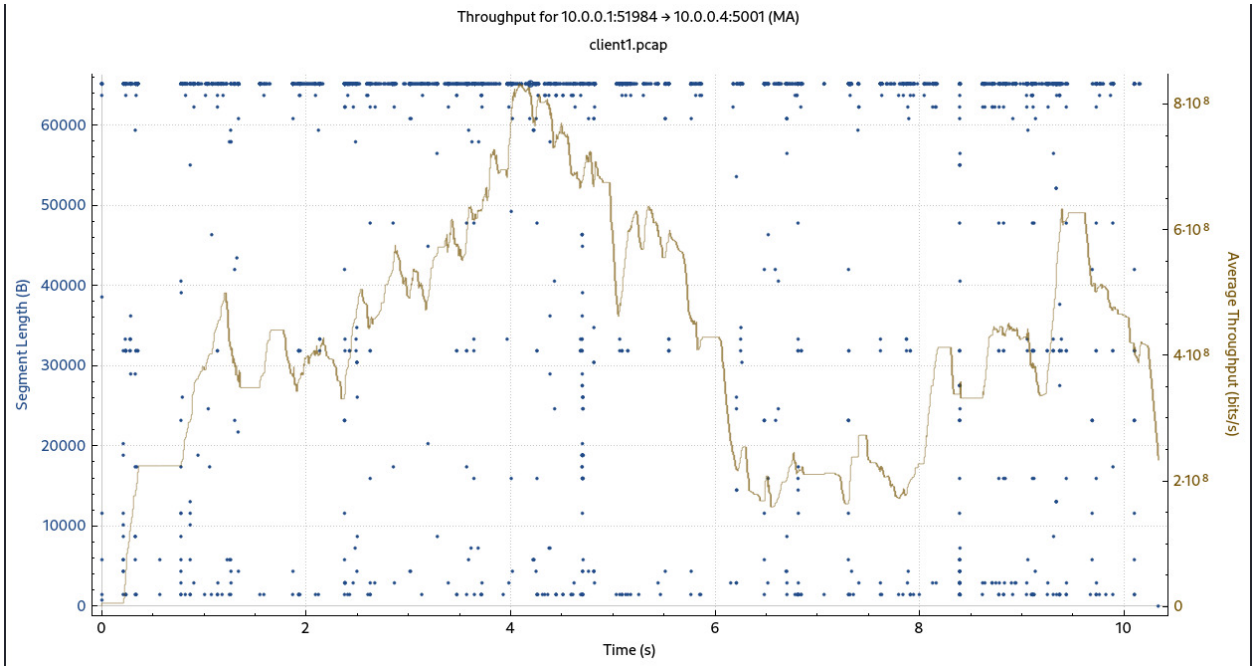

Fig 2.36 Client throughput using bbr with 3% link loss

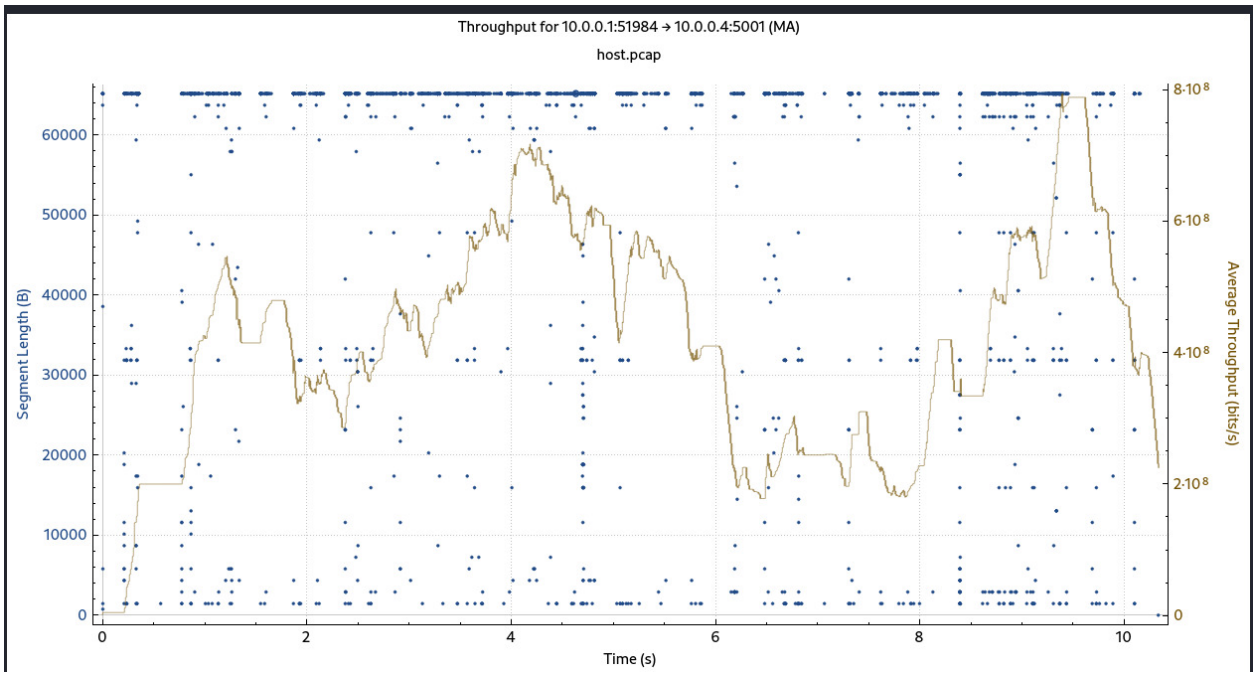

Fig 2.37 Server throughput using bbr with 3% link loss

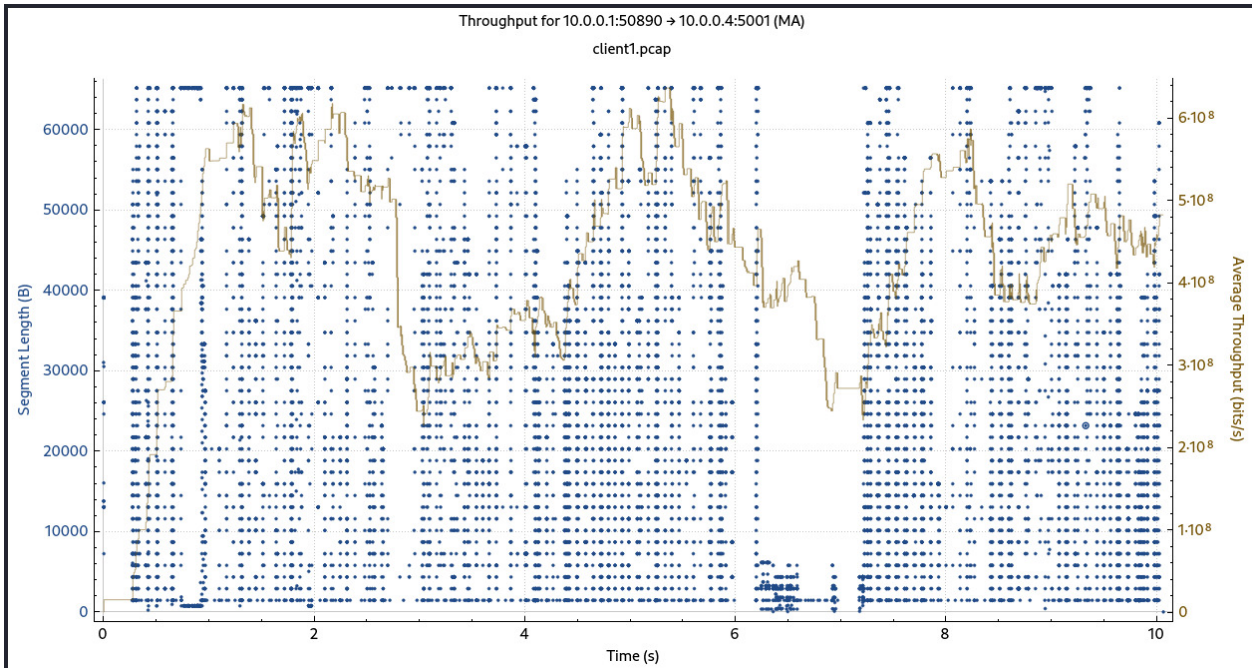

Fig 2.38 Client throughput using cubic with 1% link loss

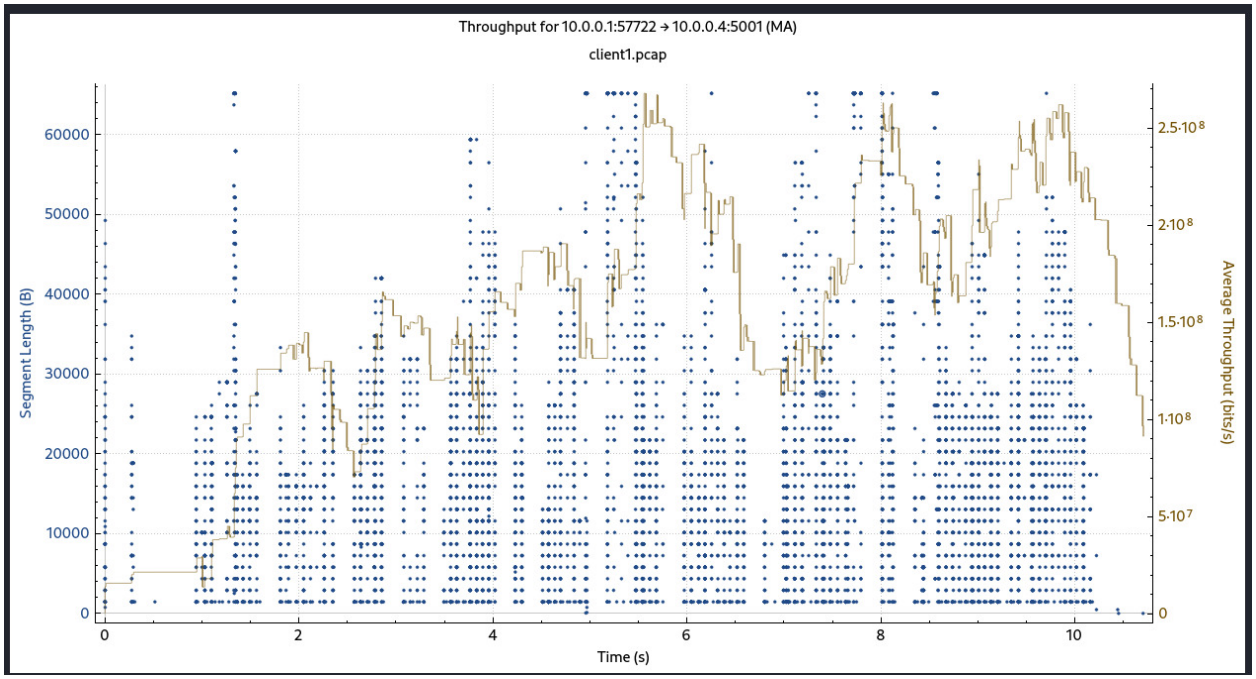

Fig 2.39 Client throughput using cubic with 3% link loss

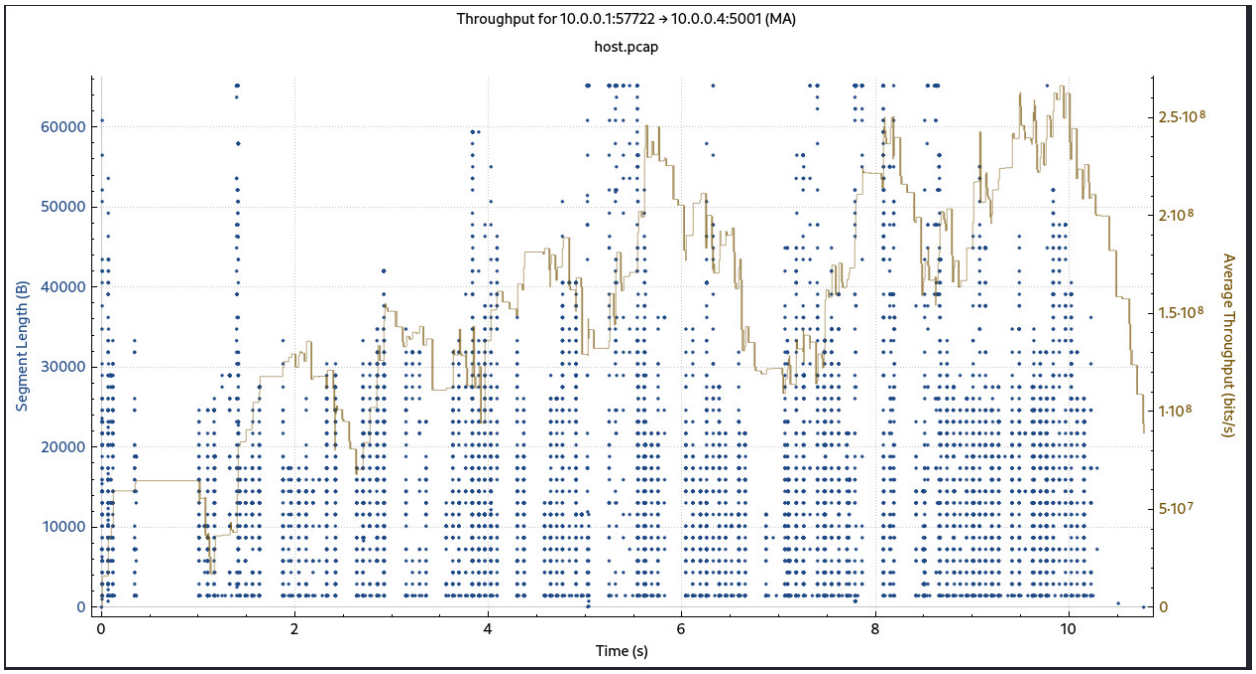

Fig 2.40 Server throughput using cubic with 3% link loss

Reasoning: One thing that we can observe in all the figures is the decrease in the number of packets, this is because because of link loss, some packets will be lost, and the clients will wait for rto time and then send the packets, also this will increase load on the server as the number of retransmissions will increase. Now from the figure we can also see the difference between the various congestion control algorithms as explained in above parts of the questions.

References:

1. https://en.wikipedia.org/wiki/TCP_Vegas
2. https://www.ietf.org/proceedings/97/slides/slides-97-iccrg-bbr-congestion-control-02.pdf
3. Stack Overflow
4. https://iperf.fr/
5. https://mininet.org/walkthrough/
6. https://intronetworks.cs.luc.edu/current/html/mininet.html
7. https://www.cisco.com/c/en/us/support/docs/dial-access/floating-static-route/118263-technote-nexthop-00.html