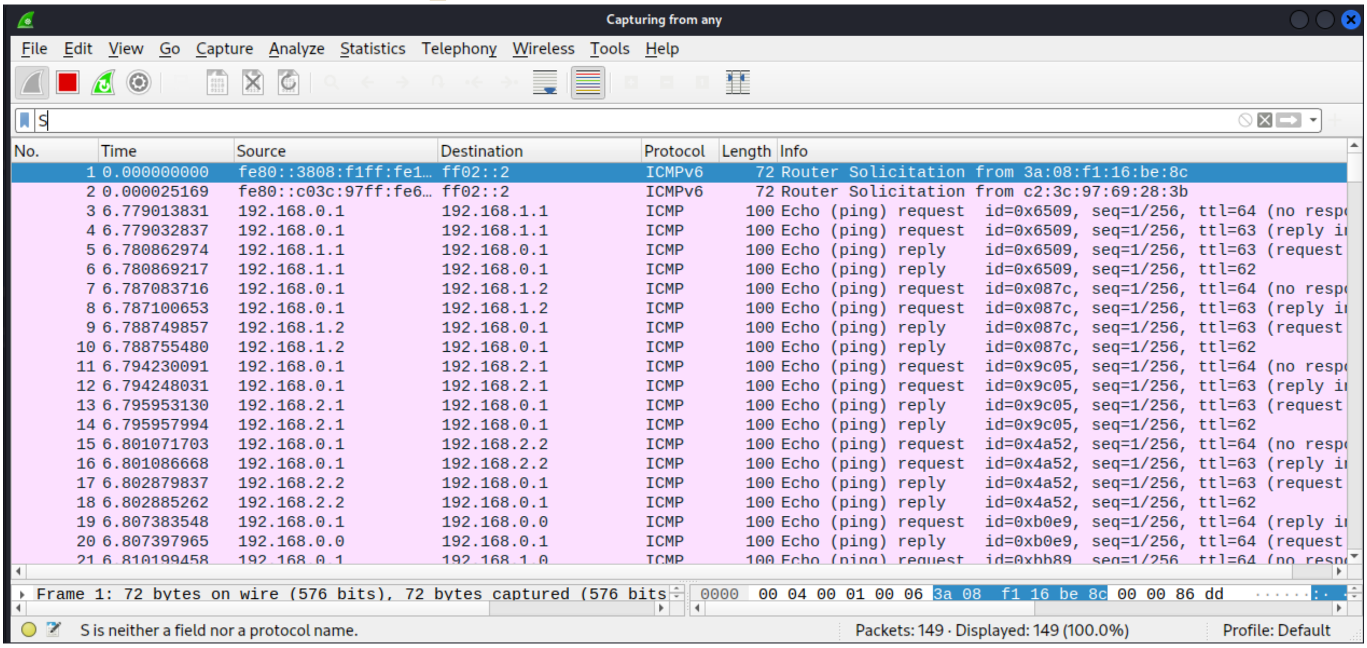
Question 1

(a) The network topology called `NetworkTopo()` has been defined inside the q1.py file using the python api of mininet. `pingall` command inside mininet give the following output.

```
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)
```

(b) Running the wireshark on router `ra` using the command `ra wireshark &`, and then running the `pingall` command inside mininet captures the following packets. The entire TCP Dump has been captured and saved inside the file called `q1_packets.pcapng`.



(c) Updated the routing table in both ra and rc so that the packets are only sent to the rb, where they are again forwarded.

> Comparing latency using the ping command and iperf tool gave the following

> 1. For path h1 -> ra -> rc -> h6
>
> ```
> mininet> h1 ping -c 5 h6
> PING 192.168.2.2 (192.168.2.2) 56(84) bytes of data.
> 64 bytes from 192.168.2.2: icmp_seq=1 ttl=62 time=6.56 ms
> 64 bytes from 192.168.2.2: icmp_seq=2 ttl=62 time=0.493 ms
> 64 bytes from 192.168.2.2: icmp_seq=3 ttl=62 time=0.143 ms
> 64 bytes from 192.168.2.2: icmp_seq=4 ttl=62 time=0.123 ms
> 64 bytes from 192.168.2.2: icmp_seq=5 ttl=62 time=0.112 ms
>
> --- 192.168.2.2 ping statistics ---
> 5 packets transmitted, 5 received, 0% packet loss, time 4067ms
> rtt min/avg/max/mdev = 0.112/1.486/6.563/2.542 ms
> ```
>
> ```
> mininet> h6 iperf -s -i 1 -p 80 &
> mininet> h1 iperf -c h6 -t 10 -i 1 -p 80
> ------------------------------------------------------------
> Client connecting to 192.168.2.2, TCP port 80
> TCP window size: 85.3 KByte (default)
> ------------------------------------------------------------
> [  1] local 192.168.0.1 port 33176 connected with 192.168.2.2 port 80 (icwnd/mss/irtt=14/1448/8773)
> [ ID] Interval       Transfer     Bandwidth
> [  1] 0.0000-1.0000 sec  5.14 GBytes  44.1 Gbits/sec
> [  1] 1.0000-2.0000 sec  5.10 GBytes  43.8 Gbits/sec
> ```
>
> 2. For the path h1 -> ra -> rb -> rc -> h6
>
> ```
> mininet> h1 ping -c 5 h6
> PING 192.168.2.2 (192.168.2.2) 56(84) bytes of data.
> 64 bytes from 192.168.2.2: icmp_seq=1 ttl=61 time=6.04 ms
> 64 bytes from 192.168.2.2: icmp_seq=2 ttl=61 time=1.19 ms
> 64 bytes from 192.168.2.2: icmp_seq=3 ttl=61 time=0.140 ms
> 64 bytes from 192.168.2.2: icmp_seq=4 ttl=61 time=0.224 ms
> 64 bytes from 192.168.2.2: icmp_seq=5 ttl=61 time=0.116 ms
>
> --- 192.168.2.2 ping statistics ---
> 5 packets transmitted, 5 received, 0% packet loss, time 4189ms
> rtt min/avg/max/mdev = 0.116/1.541/6.037/2.283 ms
> ```
>
> ```
> mininet> h6 iperf -s -i 1 -p 80 &
> mininet> h1 iperf -c h6 -t 10 -i 1 -p 80
> ------------------------------------------------------------
> Client connecting to 192.168.2.2, TCP port 80
> TCP window size: 85.3 KByte (default)
> ------------------------------------------------------------
> [  1] local 192.168.0.1 port 43302 connected with 192.168.2.2 port 80 (icwnd/mss/irtt=14/1448/10838)
> [ ID] Interval       Transfer     Bandwidth
> [  1] 0.0000-1.0000 sec  4.55 GBytes  39.1 Gbits/sec
> [  1] 1.0000-2.0000 sec  4.58 GBytes  39.3 Gbits/sec
> [  1] 2.0000-3.0000 sec  4.69 GBytes  40.3 Gbits/sec
> [  1] 3.0000-4.0000 sec  4.56 GBytes  39.1 Gbits/sec
> [  1] 4.0000-5.0000 sec  4.66 GBytes  40.1 Gbits/sec
> [  1] 5.0000-6.0000 sec  4.43 GBytes  38.1 Gbits/sec
> [  1] 6.0000-7.0000 sec  4.64 GBytes  39.8 Gbits/sec
> ```
>
> It can be observed that the latency is less for the path h1 -> ra -> rb -> rc -> h6 from both ping's `avg time` and `irtt` in iperf tool.

(d) The routing tables are printed using the code. The routing tables are in order ra, rb, rc.

> For (a),
>
> ```
> ***Routing tables***
> Kernel IP routing table
> Destination     Gateway         Genmask         Flags Metric Ref    Use Iface
> 192.168.0.0     0.0.0.0         255.255.255.0   U     0      0        0 ra-eth1
> 192.168.1.0     192.168.3.1     255.255.255.0   UG    0      0        0 ra-eth2
> 192.168.2.0     192.168.5.0     255.255.255.0   UG    0      0        0 ra-eth3
> 192.168.3.0     0.0.0.0         255.255.255.0   U     0      0        0 ra-eth2
> 192.168.5.0     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
> 192.168.0.0     192.168.3.0     255.255.255.0   UG    0      0        0 rb-eth2
> 192.168.1.0     0.0.0.0         255.255.255.0   U     0      0        0 rb-eth1
> 192.168.2.0     192.168.4.1     255.255.255.0   UG    0      0        0 rb-eth3
> 192.168.3.0     0.0.0.0         255.255.255.0   U     0      0        0 rb-eth2
> 192.168.4.0     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
> 192.168.0.0     192.168.5.1     255.255.255.0   UG    0      0        0 rc-eth3
> 192.168.1.0     192.168.4.0     255.255.255.0   UG    0      0        0 rc-eth2
> 192.168.2.0     0.0.0.0         255.255.255.0   U     0      0        0 rc-eth1
> 192.168.4.0     0.0.0.0         255.255.255.0   U     0      0        0 rc-eth2
> 192.168.5.0     0.0.0.0         255.255.255.0   U     0      0        0 rc-eth3
> ```
>
> For (c),
>
> ```
> ***Routing tables***
> Kernel IP routing table
> Destination     Gateway         Genmask         Flags Metric Ref    Use Iface
> 192.168.0.0     0.0.0.0         255.255.255.0   U     0      0        0 ra-eth1
> 192.168.1.0     192.168.3.1     255.255.255.0   UG    0      0        0 ra-eth2
> 192.168.2.0     192.168.3.1     255.255.255.0   UG    0      0        0 ra-eth2
> 192.168.3.0     0.0.0.0         255.255.255.0   U     0      0        0 ra-eth2
> 192.168.5.0     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
> 192.168.0.0     192.168.3.0     255.255.255.0   UG    0      0        0 rb-eth2
> 192.168.1.0     0.0.0.0         255.255.255.0   U     0      0        0 rb-eth1
> 192.168.2.0     192.168.4.1     255.255.255.0   UG    0      0        0 rb-eth3
> 192.168.3.0     0.0.0.0         255.255.255.0   U     0      0        0 rb-eth2
> 192.168.4.0     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
> 192.168.0.0     192.168.4.0     255.255.255.0   UG    0      0        0 rc-eth2
> 192.168.1.0     192.168.4.0     255.255.255.0   UG    0      0        0 rc-eth2
> 192.168.2.0     0.0.0.0         255.255.255.0   U     0      0        0 rc-eth1
> 192.168.4.0     0.0.0.0         255.255.255.0   U     0      0        0 rc-eth2
> 192.168.5.0     0.0.0.0         255.255.255.0   U     0      0        0 rc-eth3
> ```
>
> The routing tables have also been saved to the files `q1_routes_a.txt` and `q1_routes_c.txt` for the respective questions (a) and (c)

Question 2

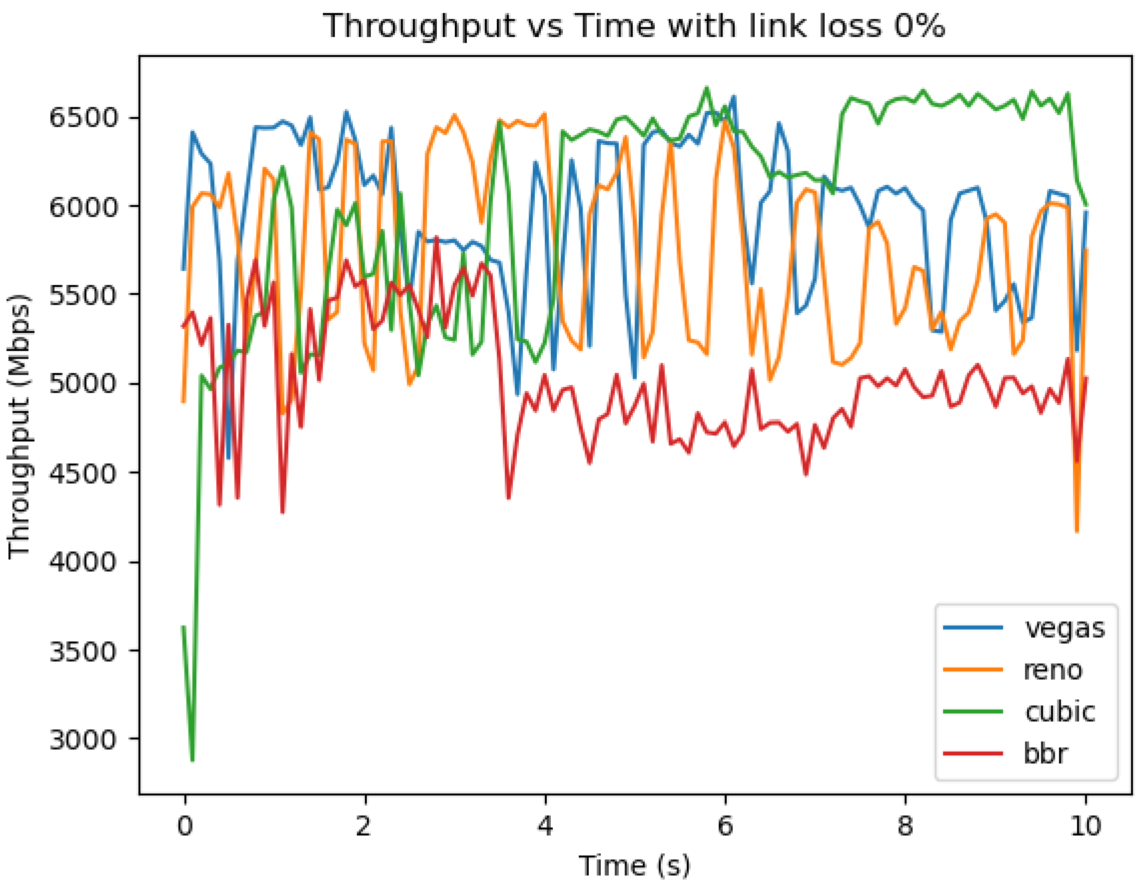
(a) The network topology given in the image is defined in `NetworkTopo()` class, and the class takes the link*loss parameter, which sets the value of the _s1-s2* link packet loss percentage. The file `q2.py` can be run using python with `sudo` permissions. The file can be run using the below command. Where the values of config(required) can be `b` (or) `c`, values of congestion control(not required) can be any one of `{Vegas, Cubic, Reno, BBR}`, and values of link loss(not required) can be any integer between `0` and `100`

```
python q2.py --config[-c] <config> --link-loss[-ll] <link loss> --
congestion-control[-cc] <congestion control>
```

> ***Note:***
>
> 1. If no `congestion-control` option is given for config `b` then the program plots Throughput for all values of all the above given congestion control mechanism, for the client `h1`.
> 2. If a `congestion-control` option is given for config `c` then the program plots Throughput for all hosts `h1`, `h2`, and `h3`, for the given value of congestion control in the same graph. If it is not given then it plots Throughput for all values of the congestion control mechanism for each given host {`h1`, `h2`, `h3`} in a different graph.
> 3. If no `link-loss` option is mentioned then the throughput analysis is done using `0` link loss for `s1-s2` link.
> 4. The program does not enter into `mininet` CLI.

(b) The client is run for `10sec` and the plots are obtained by running the file using the below command.

```
sudo python q2.py --config b
```

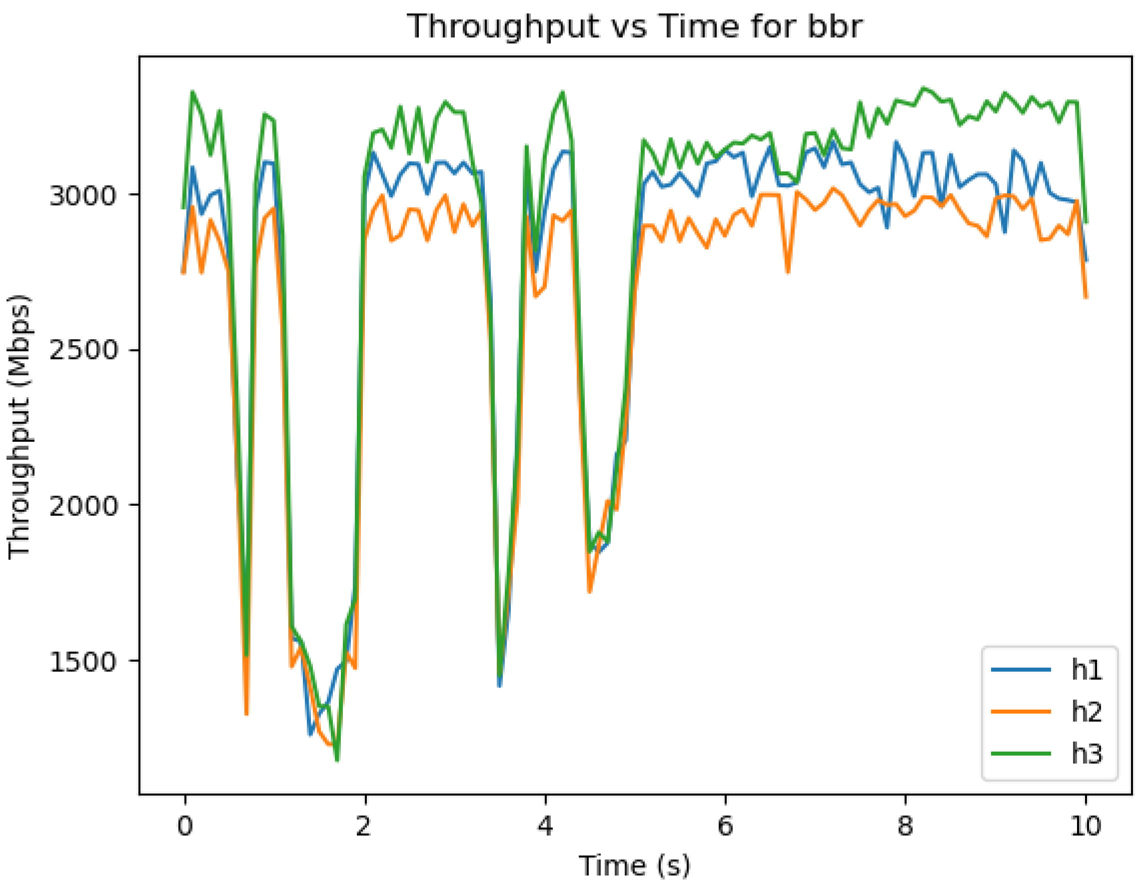


It can be observed that BBR congestion control gives less throughput, vegas shows a steady throughput after some time. Cubic, and Reno congestion control shows a decreasing throughput after some time, and cubic also increases after the decrease. Reno and BBR have slow start, whereas Vegas and Cubic have fast start.

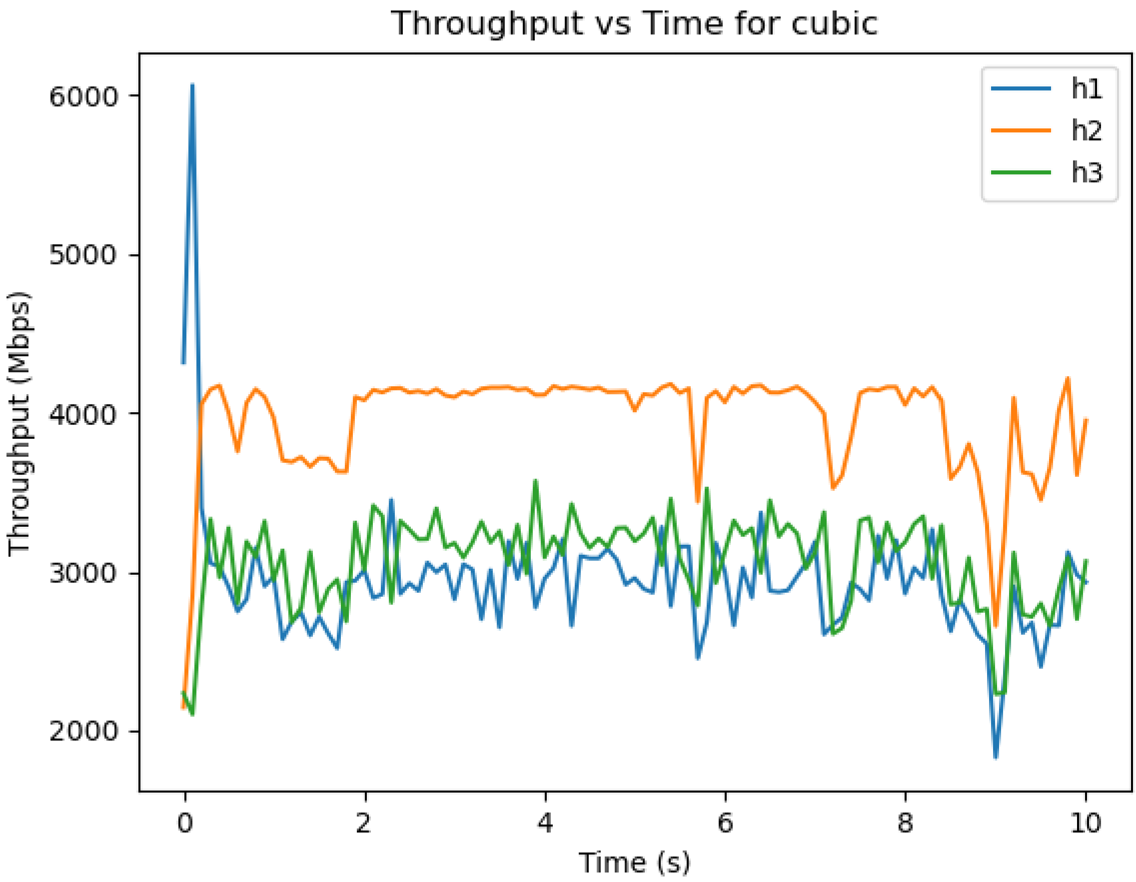
(c) The following commands were used to get the throughput plots for different hosts using the mentioned congestion control algorithm.

> ***Note:*** File when run at different times might produce different plots depending on which host first starts sending packets.

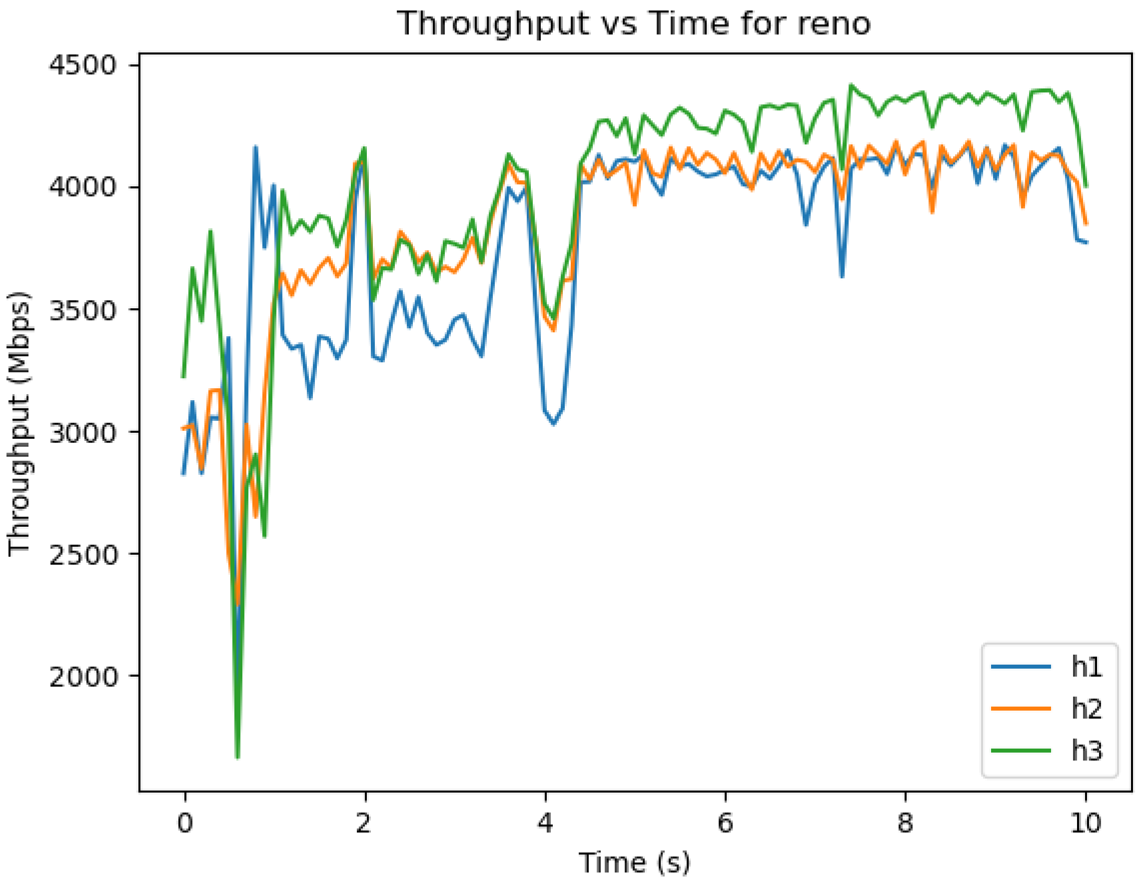
```
sudo python q2.py --config c -cc BBR
```



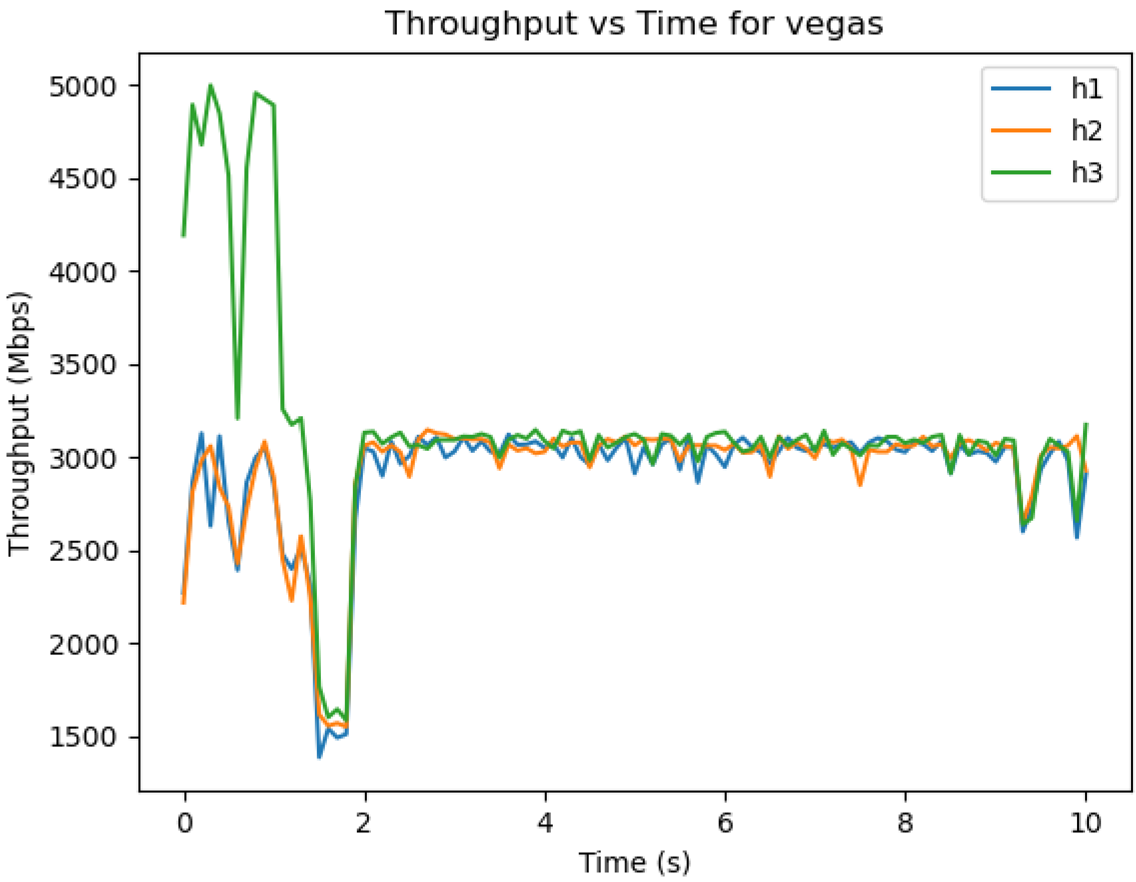

```
sudo python q2.py --config c -cc Cubic
```




```
sudo python q2.py --config c -cc Reno
```




```
sudo python q2.py --config c -cc Vegas
```

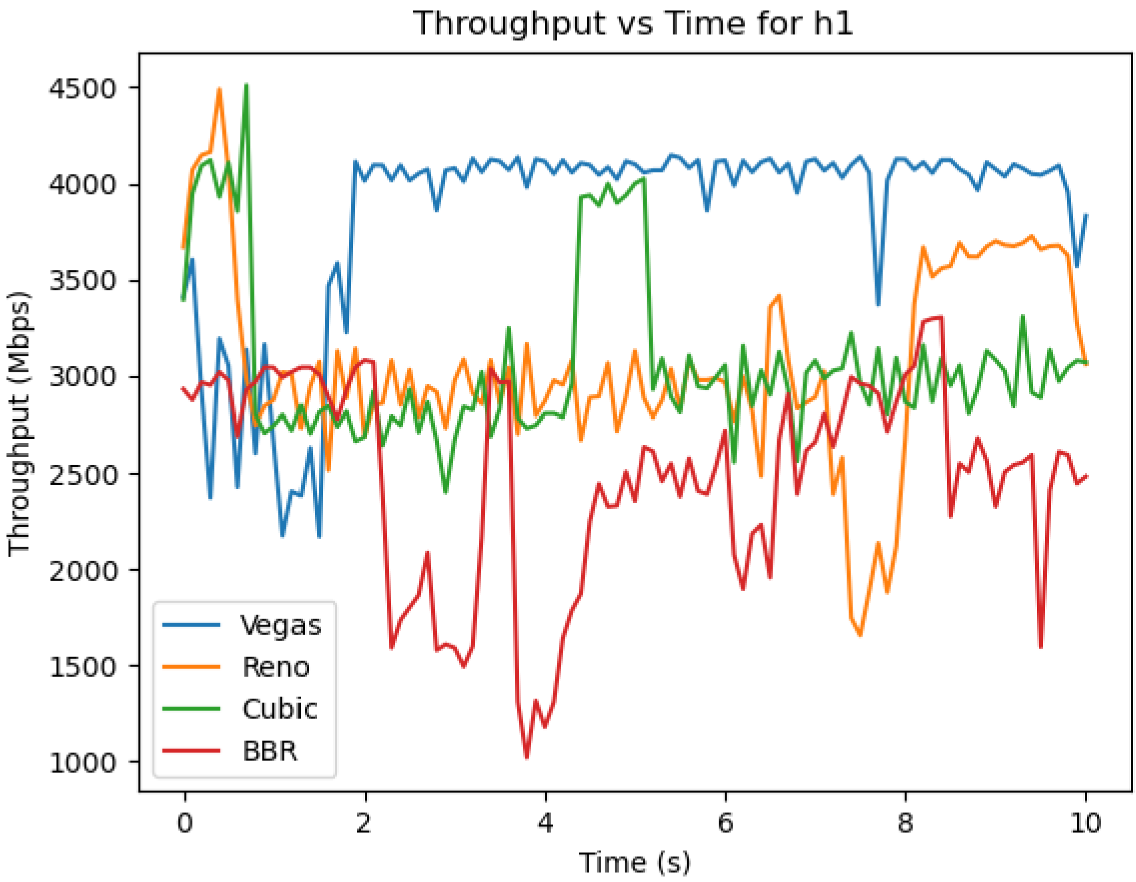


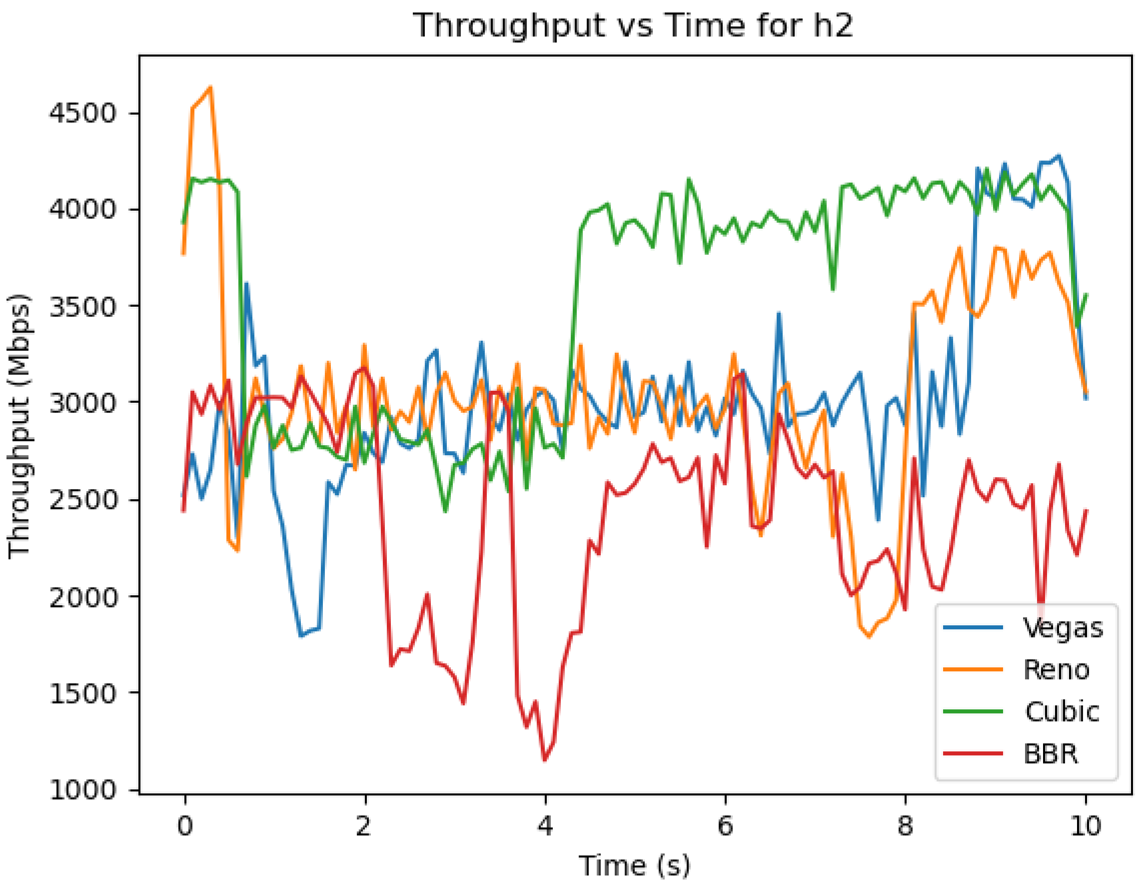
It can be observed that BBR congestion control gives less throughput, vegas appears to be fair for different hosts. BBR, and Reno congestion control show frequent changes. In case of BBR, it appears that all the hosts sharing the bandwidth respond similarly to the congestion control algorithm. Vegas appears to be the first to start fair sharing of bandwidth among the hosts, and it does not to show frequent changes in throughput if the network conditions are not changed. This may be because of the fact that Vegas uses RTT to throttle bandwith usage, and it does not use packet loss as a metric. Cubic and Reno use packet loss as a metric to throttle the andwidth usage, and hence they show frequent changes in throughput.

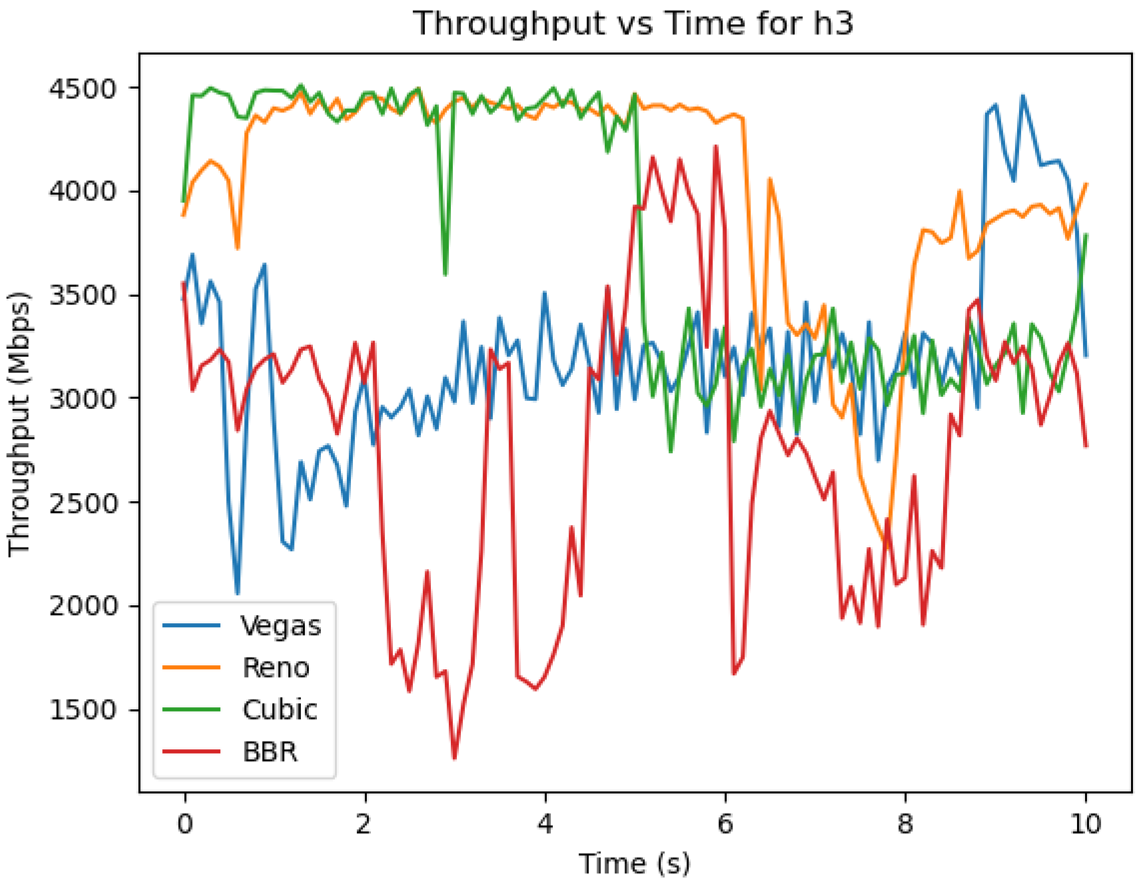
From questions (b) and (c) it can be observed that both network conditions and congestion control algorithm affect the throughput.

The following command was used to get the throughput plots for different congestion control algorithm for a given host.

```
sudo python q2.py --config c
```

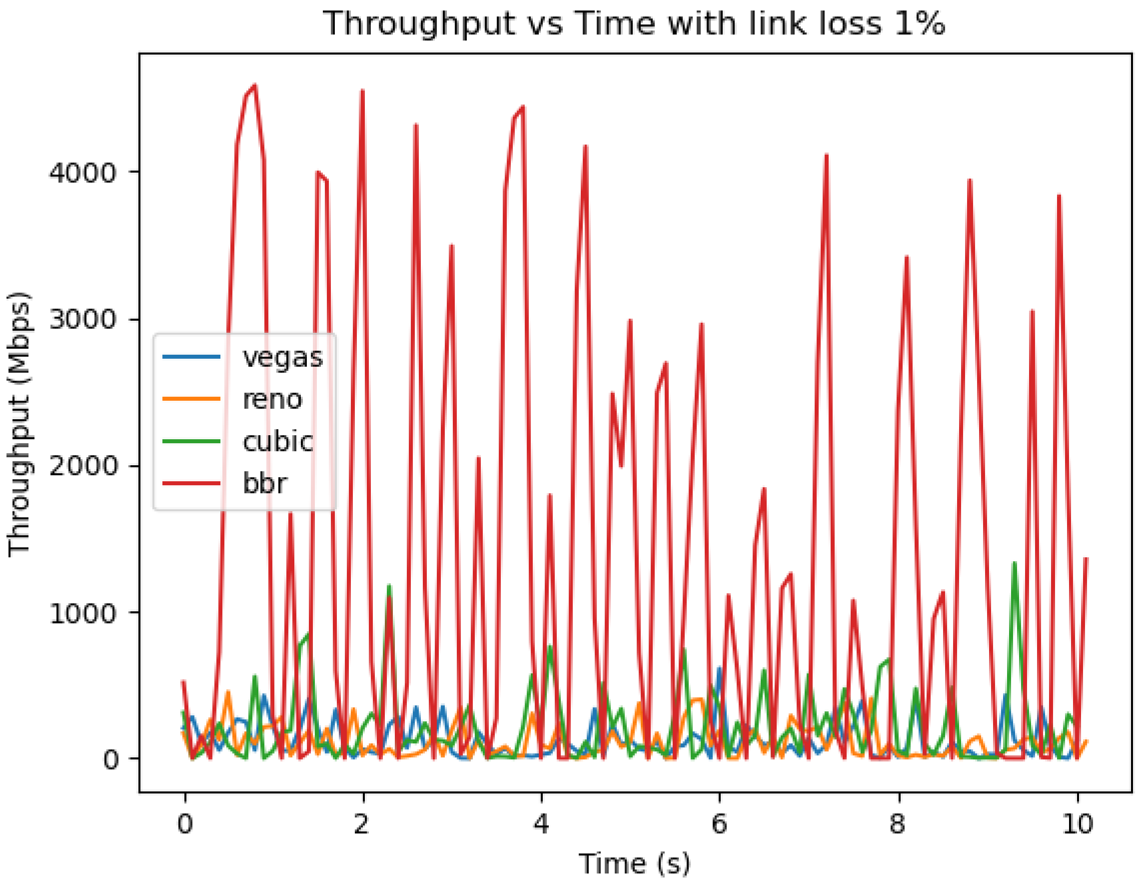

Throughput vs Time for h1
Throughput (Mbps)
4500
4000
3500
3000
2500
2000
1500
1000
Time (s)
0
2
4
6
8
10
Vegas
Reno
Cubic
BBR


Throughput vs Time for h2
Throughput (Mbps)
4500
4000
3500
3000
2500
2000
1500
1000
Time (s)
0
2
4
6
8
10
Vegas
Reno
Cubic
BBR

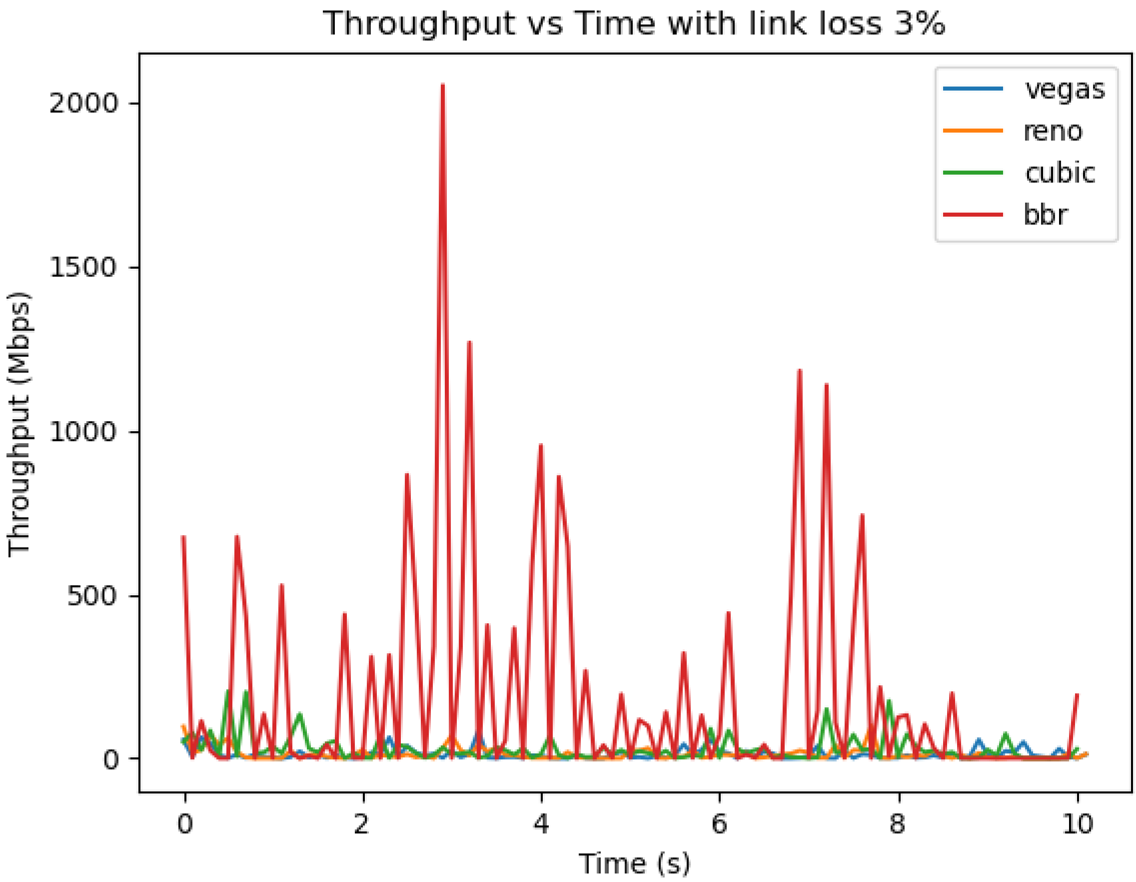


(d) The following command was used to get the throughput plots on client side for different congestion control algorithm for a given link loss{1, 3}.

```
sudo python q2.py --config b -ll 1
```




```
sudo python q2.py --config b -ll 3
```



It appears that BBR shows the highest throughput in case of packets losses and very high fluctuations in throughput during loss as compared to others. Vegas shows very less throughput change. This might be becuase of the fact that vegas uses RTT to throttle bandwith usage, and it does not use packet loss as a metric. From this we can say that BBR causes more congestion in the network as compared to others(which are sensitive to congestion in the network).

References:

1. Iperf
2. Mininet
3. Mininet Python API
4. Mininet Router Implementation
   5 .Congestion Control Algorithms