# CS 433 Computer Networks

Write Up

# Assignment-2

Karan Bhardwaj:20110093
Manpreet Singh:20110109

---

# Q1

### a. Implementation

Here we have implemented the given topology using three subnets each having two hosts and a router.
The router has been implemented as a combination of a switch and a host.
The IP addresses are as:-

ra :100.101.1.1
rb:100.101.0.1
rc:100.103.0.1
h1:100.101.1.100
h2:100.101.1.101
h3:100.102.0.100
h4:100.102.0.101
h5:100.103.0.100
h6:100.103.0.101

**Command Used:** sudo mn -c; python3 Q1.py

Mininet CLI will open, with the custom topology designed.
Now, using the **pingall** command we have tested the connections between all hosts and routers and as per the results the connections are working well. We can see the topology is working well as 100 % of the packets sent are received.

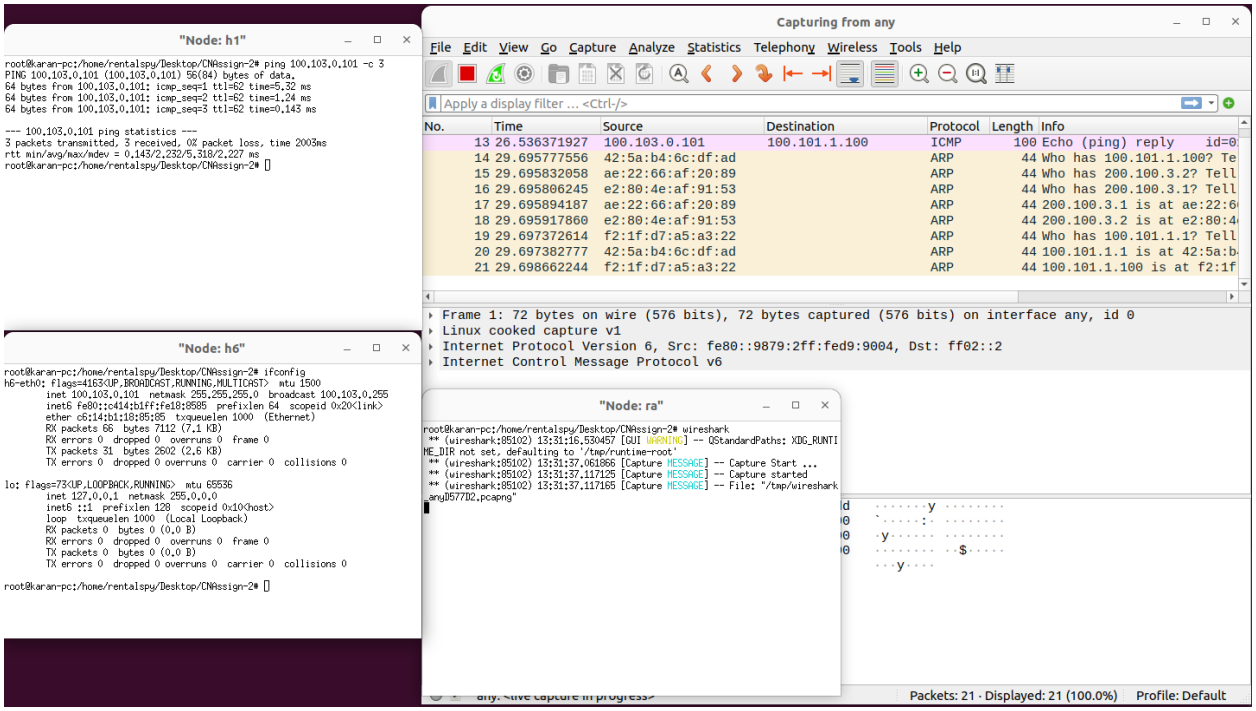
```
*** Starting CLI:
mininet> pingall
*** Ping: testing ping reachability
h1 -> h2 h3 h4 h5 h6 ra rb rc
h2 -> h1 h3 h4 h5 h6 ra rb rc
h3 -> h1 h2 h4 h5 h6 ra rb rc
h4 -> h1 h2 h3 h5 h6 ra rb rc
h5 -> h1 h2 h3 h4 h6 ra rb rc
h6 -> h1 h2 h3 h4 h5 ra rb rc
ra -> h1 h2 h3 h4 h5 h6 rb rc
rb -> h1 h2 h3 h4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

## b. Wireshark

Below shown snippets show the working of wireshark packet capture for the route h1->$r_a$->$r_c$->h6.

**Wireshark** was open on ra router using **xterm**, to capture the packets through it.

Here we have sent three packets to h6 from h1 using command **ping 100.103.0.101 -c 3** we observe the time taken for the packet to reach its destination.

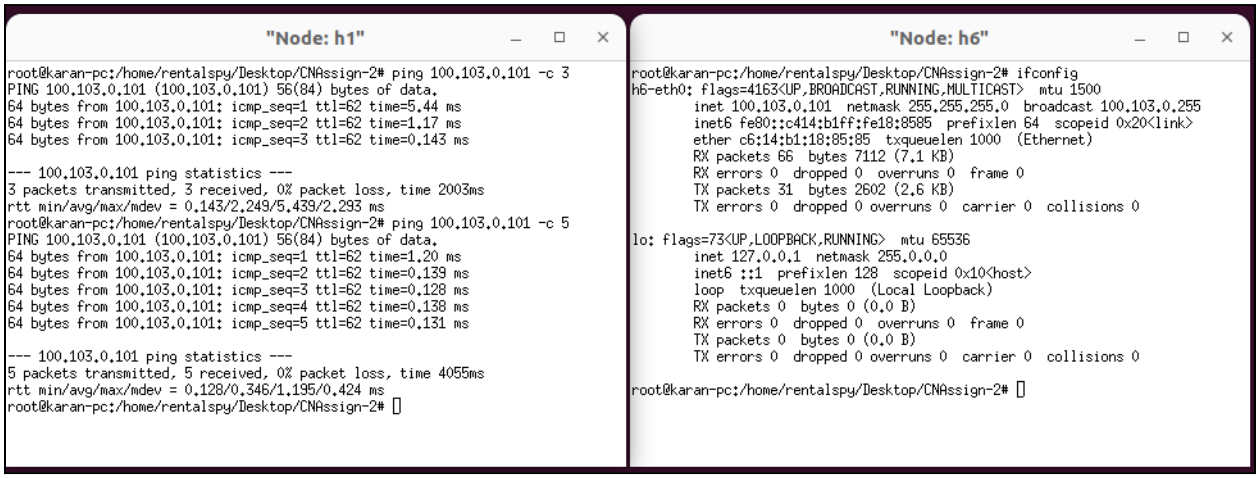


### c. Latency

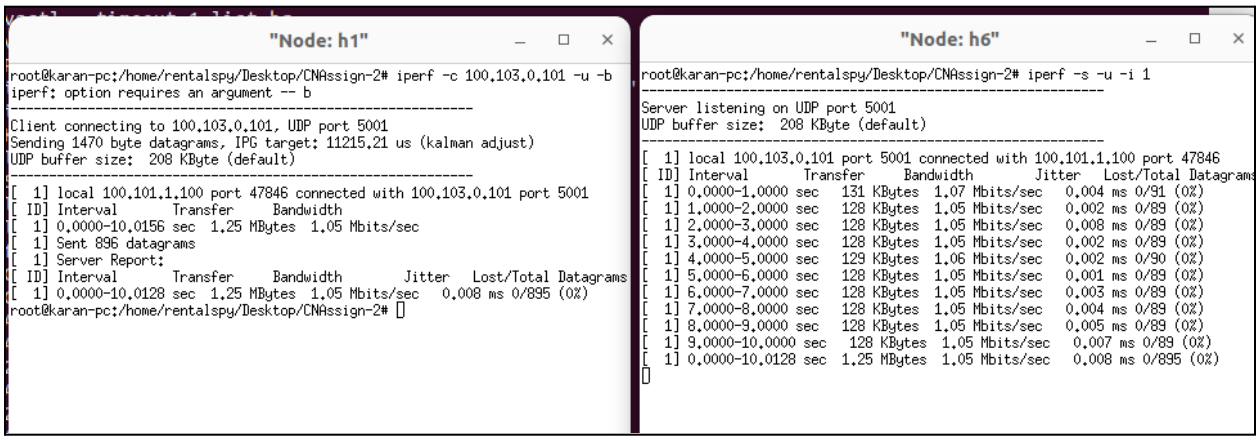
Below are the output snippets for the default route from host h1 to h6 that is **h1->$r_a$->$r_c$->h6**. Here we are first sending 5 packets from h1 to h6. In the image shown below the RTT corresponding to each of the sent packets is shown here.

## Using ping command:

Avg RTT= 0.346ms.



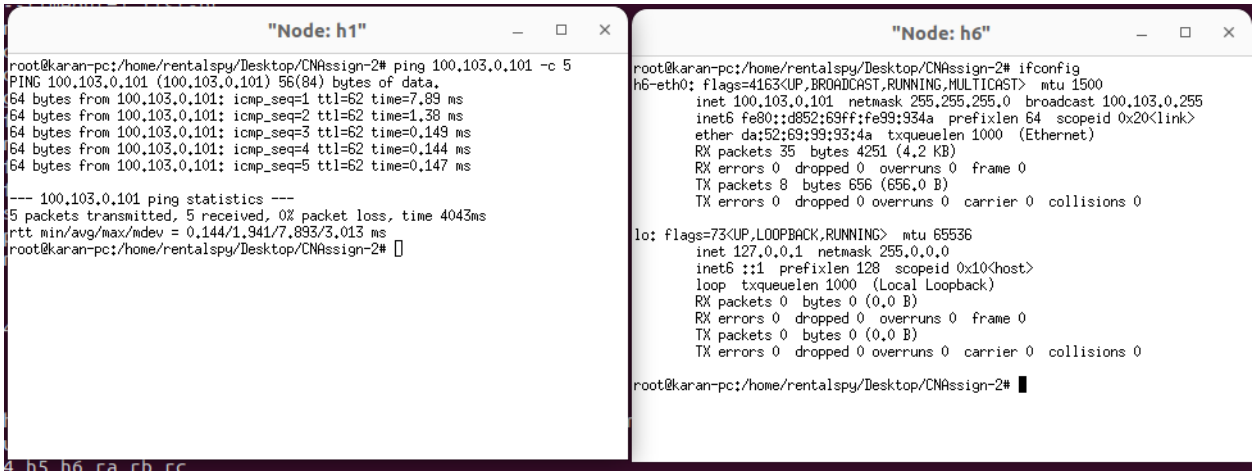

## Using iperf command:



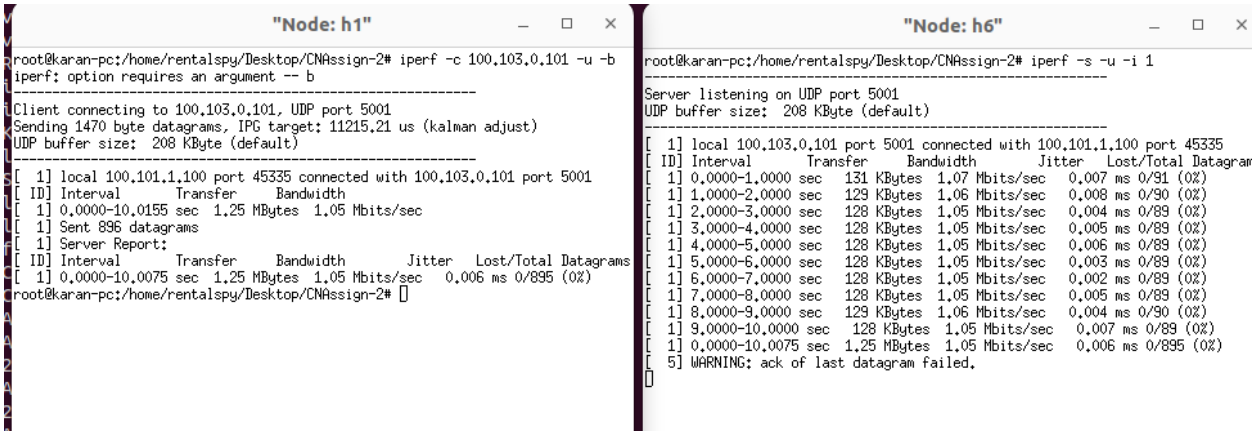
Below are the output snippets for the new route from host h1 to h6 that is **h1->$r_a$->$r_b$->$r_c$->h6**. Here also we are sending 5 packets from h1 to h6. The image below shows the RTT corresponding to each of the packets received.

## Using ping command:

Avg RTT = 1.941ms




## Using iperf command:




**Observation:** The latency is higher for the longer path **h1->$r_a$->$r_b$->$r_c$->h6.**

d. Routing table

Topology with route **h1->$r_a$->$r_c$->h6**.

```
*** Adding links:
(h1, s1) (h2, s1) (h3, s2) (h4, s2) (h5, s3) (h6, s3) (ra, rb) (ra, rc) (rb, rc) (
*** Configuring hosts
h1 h2 h3 h4 h5 h6 ra rb rc
*** Starting controller
c0
*** Starting 3 switches
s1 s2 s3 ...
*** Waiting for switches to connect
s1 s2 s3
*** Adding static routes on routers:
*** Routing Tables on Routers:
Kernel IP routing table
Destination     Gateway         Genmask         Flags Metric Ref    Use Iface
100.101.1.0     0.0.0.0         255.255.255.0   U     0      0        0 ra-eth1
100.102.0.0     200.100.1.2     255.255.255.0   UG    0      0        0 l
100.103.0.0     200.100.3.2     255.255.255.0   UG    0      0        0 p
200.100.1.0     0.0.0.0         255.255.255.0   U     0      0        0 l
200.100.3.0     0.0.0.0         255.255.255.0   U     0      0        0 p
Kernel IP routing table
Destination     Gateway         Genmask         Flags Metric Ref    Use Iface
100.101.1.0     200.100.1.1     255.255.255.0   UG    0      0        0 m
100.102.0.0     0.0.0.0         255.255.255.0   U     0      0        0 rb-eth1
100.103.0.0     200.100.2.2     255.255.255.0   UG    0      0        0 n
200.100.1.0     0.0.0.0         255.255.255.0   U     0      0        0 m
200.100.2.0     0.0.0.0         255.255.255.0   U     0      0        0 n
Kernel IP routing table
Destination     Gateway         Genmask         Flags Metric Ref    Use Iface
100.101.1.0     200.100.3.1     255.255.255.0   UG    0      0        0 q
100.102.0.0     200.100.2.1     255.255.255.0   UG    0      0        0 o
100.103.0.0     0.0.0.0         255.255.255.0   U     0      0        0 rc-eth1
200.100.2.0     0.0.0.0         255.255.255.0   U     0      0        0 o
200.100.3.0     0.0.0.0         255.255.255.0   U     0      0        0 q
*** Starting CLI:
mininet> pingall
```

Now, applying **pingall** with changes made

```
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 X X rb X
rb -> h1 h2 h3 h4 h5 h6 ra rc
rc -> h1 h2 h3 h4 h5 h6 ra rb
*** Results: 4% dropped (69/72 received)
mininet>
```

There was a 4% drop.

Topology with route **h1->$r_a$->$r_b$->$r_c$->h6**.

```
*** Starting 3 switches
s1 s2 s3 ...
*** Waiting for switches to connect
s1 s2 s3
*** Adding static routes on routers:
*** Routing Tables on Routers:
Kernel IP routing table
Destination     Gateway         Genmask         Flags Metric Ref    Use Iface
100.101.1.0     0.0.0.0         255.255.255.0   U     0      0        0 ra-eth1
100.102.0.0     200.100.1.2     255.255.255.0   UG    0      0        0 l
100.103.0.0     200.100.1.2     255.255.255.0   UG    0      0        0 l
200.100.1.0     0.0.0.0         255.255.255.0   U     0      0        0 l
200.100.3.0     0.0.0.0         255.255.255.0   U     0      0        0 p
Kernel IP routing table
Destination     Gateway         Genmask         Flags Metric Ref    Use Iface
100.101.1.0     200.100.1.1     255.255.255.0   UG    0      0        0 m
100.102.0.0     0.0.0.0         255.255.255.0   U     0      0        0 rb-eth1
100.103.0.0     200.100.2.2     255.255.255.0   UG    0      0        0 n
200.100.1.0     0.0.0.0         255.255.255.0   U     0      0        0 m
200.100.2.0     0.0.0.0         255.255.255.0   U     0      0        0 n
Kernel IP routing table
Destination     Gateway         Genmask         Flags Metric Ref    Use Iface
100.101.1.0     200.100.3.1     255.255.255.0   UG    0      0        0 q
100.102.0.0     200.100.2.1     255.255.255.0   UG    0      0        0 o
100.103.0.0     0.0.0.0         255.255.255.0   U     0      0        0 rc-eth1
200.100.2.0     0.0.0.0         255.255.255.0   U     0      0        0 o
200.100.3.0     0.0.0.0         255.255.255.0   U     0      0        0 q
*** Starting CLI:
mininet>
```

# Q2

The IP addresses as:

h1:10.0.0.1
h2:10.0.0.2
h3:10.0.0.3
h4:10.0.0.4

```
class Mytopo(Topo):
    def build(self, **_opts):
        s1 = self.addSwitch('s1')
        s2 = self.addSwitch('s2')
        h1 = self.addHost('h1')
        h2 = self.addHost('h2')
        h3 = self.addHost('h3')
        h4 = self.addHost('h4')
        self.addLink(h1, s1)
        self.addLink(h2, s1)
        self.addLink(h3, s2)
        self.addLink(h4, s2)
        self.addLink(s1, s2)
```

## a. TCP Client-Server Program

Created two files server.py and client.py. The host h1, h2 and h3 will run client.py and will send a message to the server h4 by creating a TCP socket. The server h4 will be running server.py. It will receive the message. All the sent and received messages are stored in a txt.file. This is done to check whether the client-server connection is working properly or not

**Command Used:** sudo mn -c; python3 Q2.py

**Client.py**

```
import socket

client_socket = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
client_socket.connect(('10.0.0.4', 8888))  # Assuming server IP is 10.0.0.4
message = 'Hello from the client'
client_socket.send(message.encode())
client_socket.close()

# Save the message to a file
with open('messages.txt', 'a') as file:
    file.write(f'Client: {message}\n')
```

**Server.py**

```
import socket

server_socket = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
server_socket.bind(('0.0.0.0', 8888))
server_socket.listen(1)
print('Server listening on port 8888')

while True:
    connection, address = server_socket.accept()
    data = connection.recv(1024).decode()
    print(f'Received data from {address}: {data}')

    # Save the message to a file
    with open('messages.txt', 'a') as file:
        file.write(f'Server message received from ({address}): {data}\n')

    connection.close()
```

```
Client: Hello from the client
Client: Hello from the client
Server message received from (('10.0.0.1', 53858)): Hello from the client
Server message received from (('10.0.0.2', 42112)): Hello from the client
Client: Hello from the client
Server message received from (('10.0.0.3', 44278)): Hello from the client
```

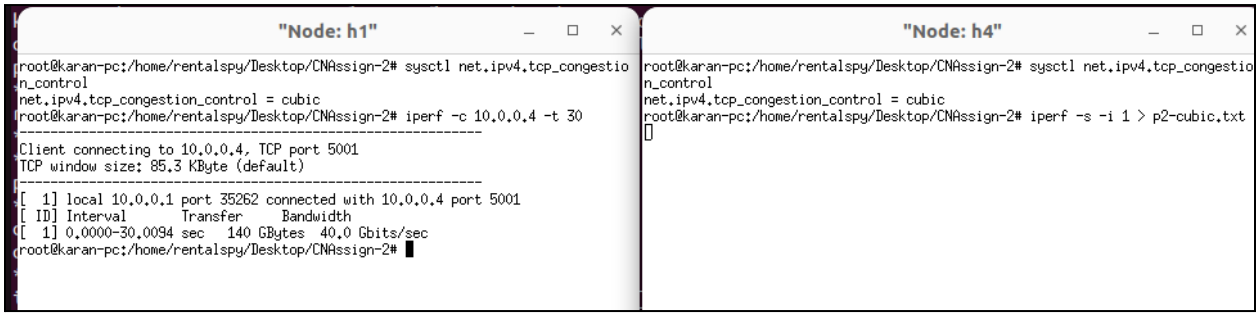
## b. Client- h1, Server- h4

Running the iperf server on host h4 and iperf client on host h1 simultaneously for 30 seconds.

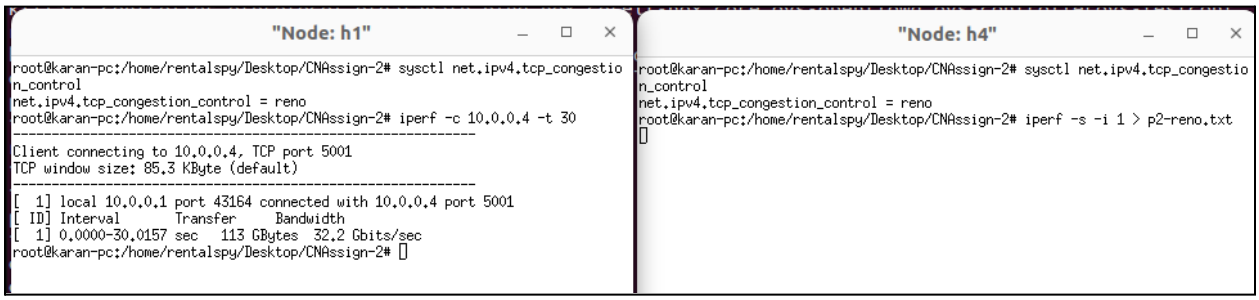
**Note:** Output of xterm terminals are provided as a confirmation that congestion has been applied. These outputs are generated manually, although the code is automated.

The command on execution generates a txt file containing the Throughput information along with bandwidth.

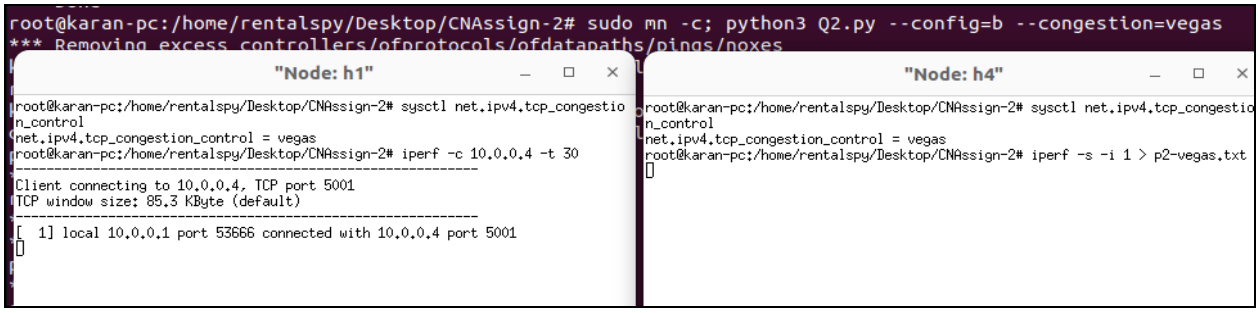
Congestion control mechanism = cubic
**Command:** sudo mn -c; python3 Q2.py --config=b --congestion=cubic
**Avg.Throughput:** 40.0 Gbps




Congestion control mechanism = reno
**Command:** sudo mn -c; python3 Q2.py --config=b --congestion=reno
**Avg.Throughput:** 32.2 Gbps



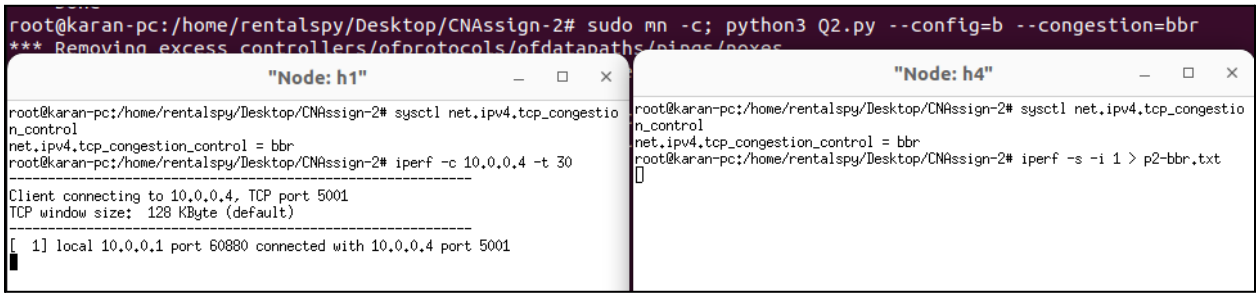

Congestion control mechanism = vegas
**Command:** sudo mn -c; python3 Q2.py --config=b --congestion=vegas
**Avg.Throughput:**38.3 Gbps



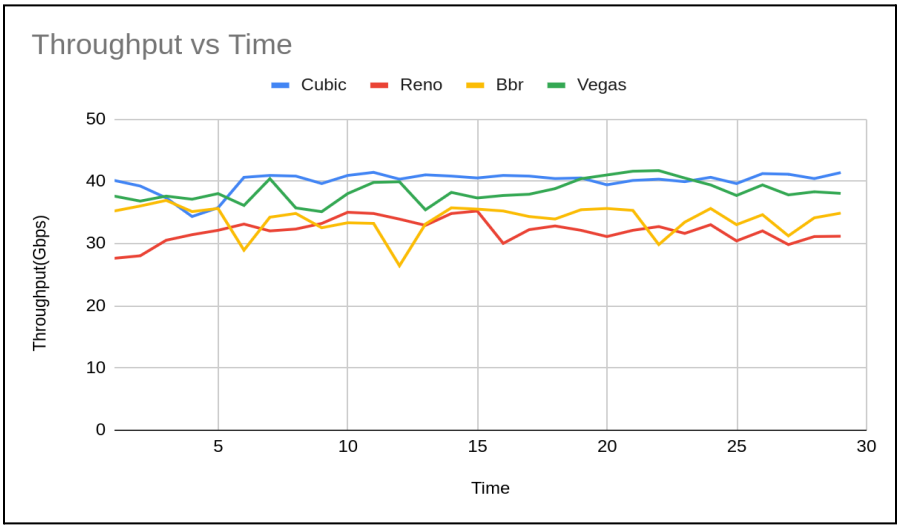

Congestion control mechanism = bbr

**Command:** sudo mn -c; python3 Q2.py --config=b --congestion=bbr

**Avg.Throughput:** 33.99 Gbps




# Graph




**Avg. Throughput Order:** Cubic > Vegas > Bbr > Reno

Since, there is only connection h1->h4, there is no competition for available bandwidth with any connection. So, we can easily see the effect of different congestion schemes.

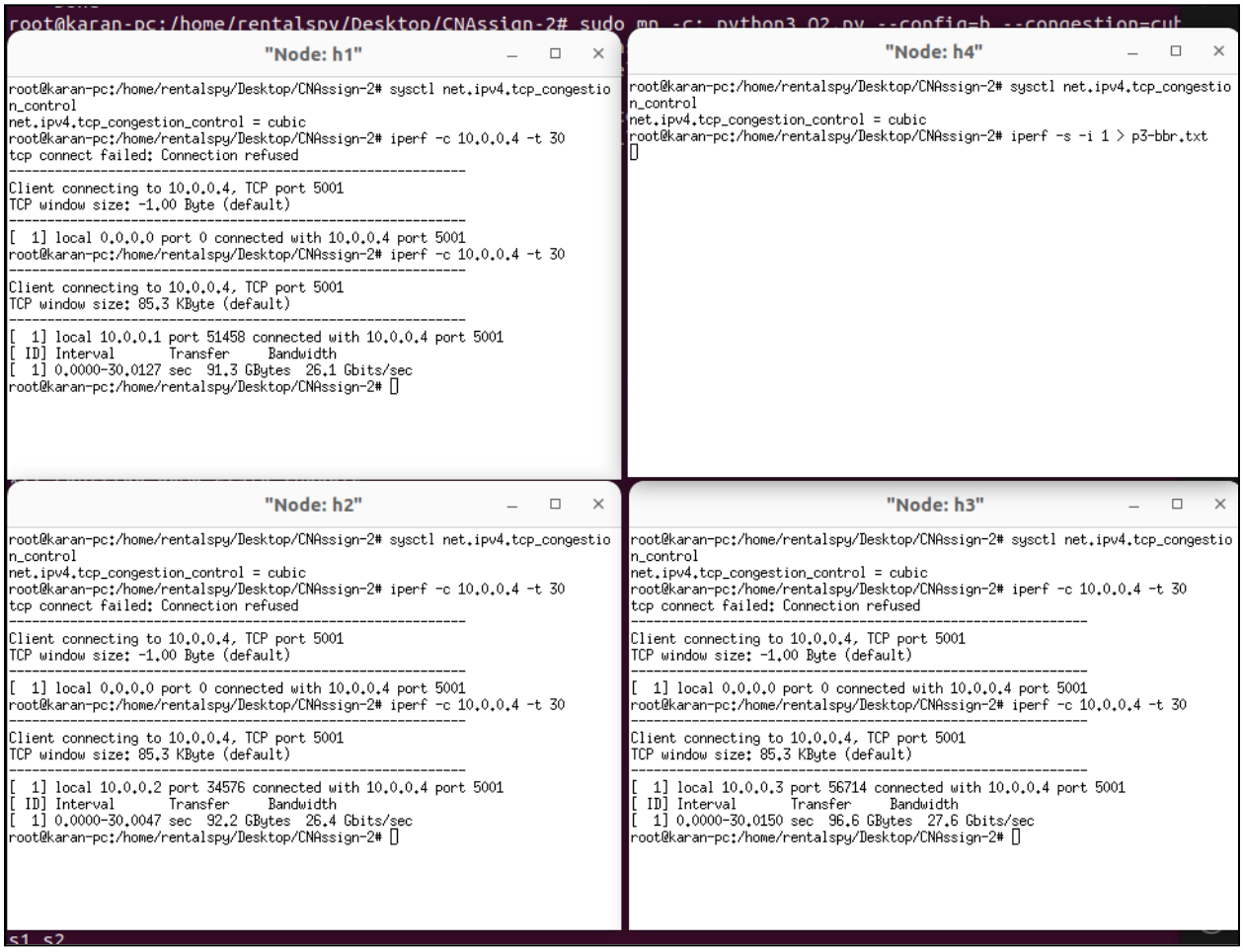
## c. Client - h1, h2, h3 , Server- h4

Running the iperf server on host h4 and iperf client on host h1, h2 and h3 simultaneously for 30 seconds.

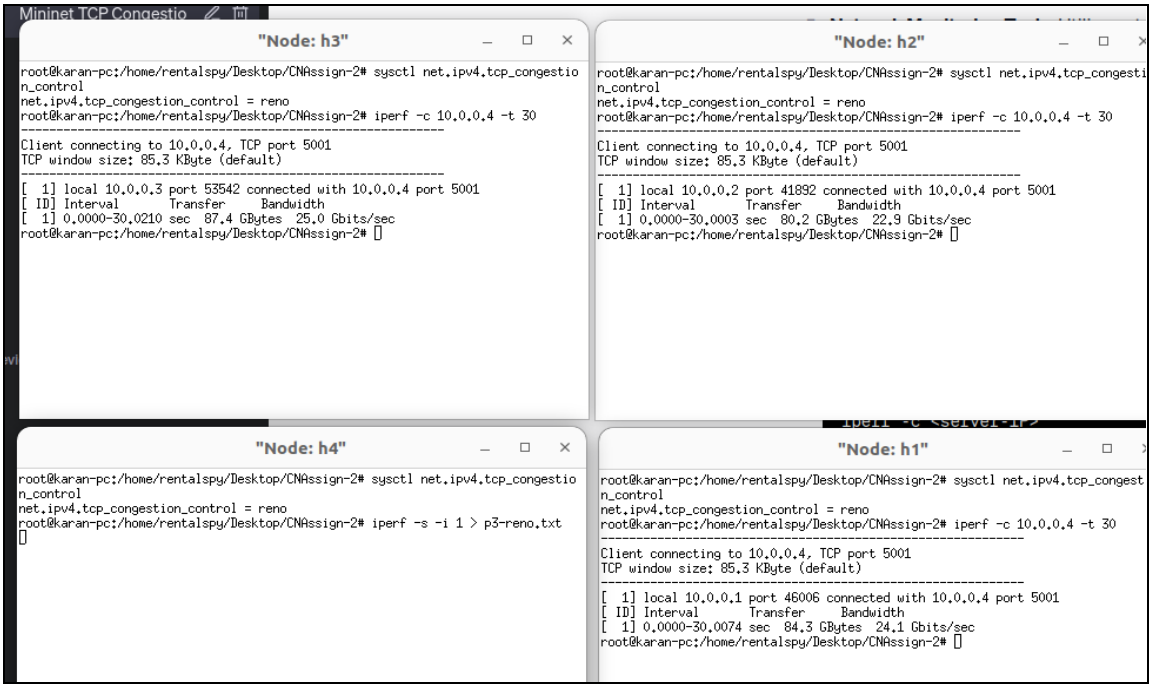
**Note:** Output of xterm terminals are provided as a confirmation that congestion has been applied. These outputs are generated manually, although the code is automated.

The command on execution generates a txt file containing the Throughput information along with bandwidth.

Congestion control mechanism = cubic
**Command:** sudo mn -c; python3 Q2.py --config=c --congestion=cubic

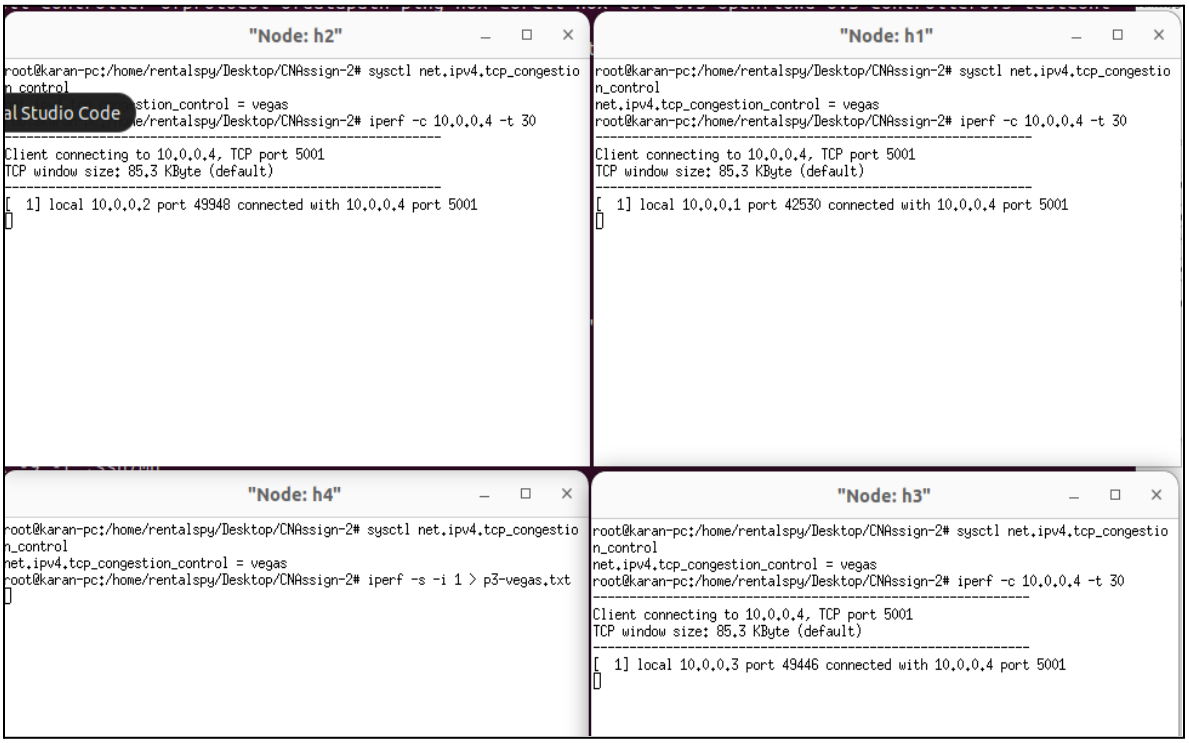



Congestion control mechanism = reno
**Command:** sudo mn -c; python3 Q2.py --config=c --congestion=reno



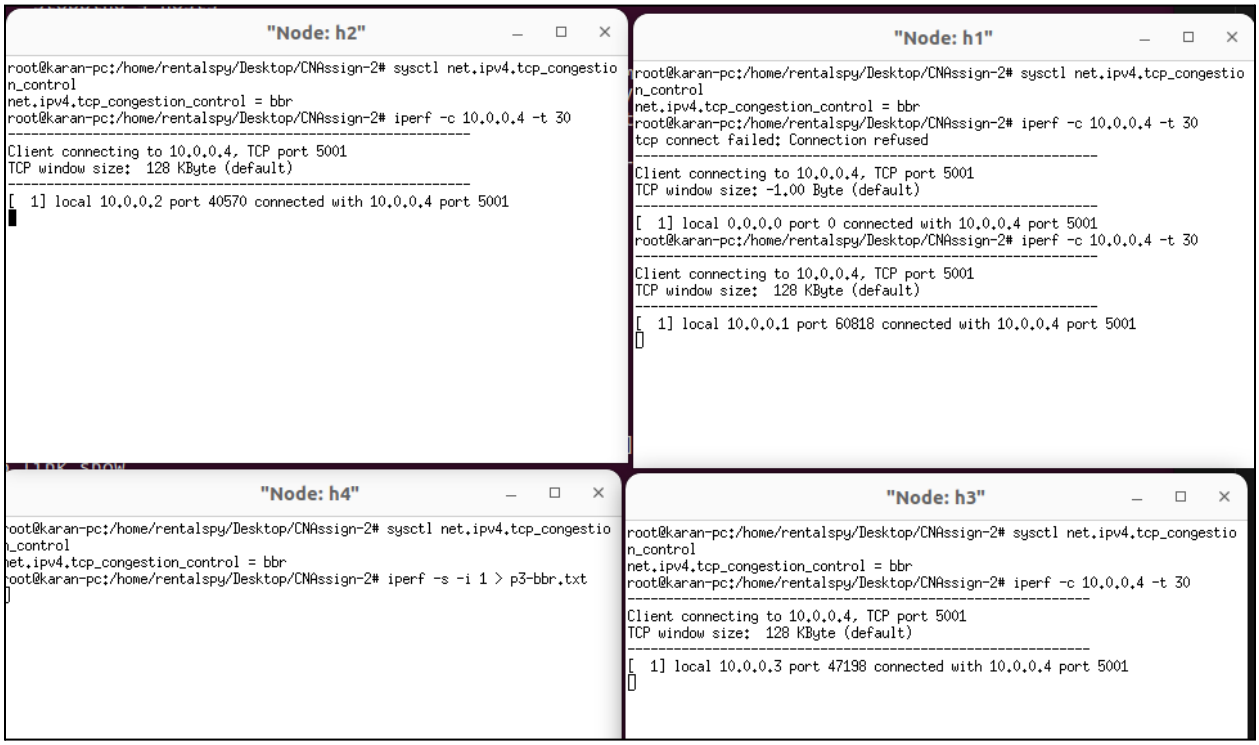

Congestion control mechanism = vegas

**Command:** sudo mn -c; python3 Q2.py --config=c --congestion=vegas

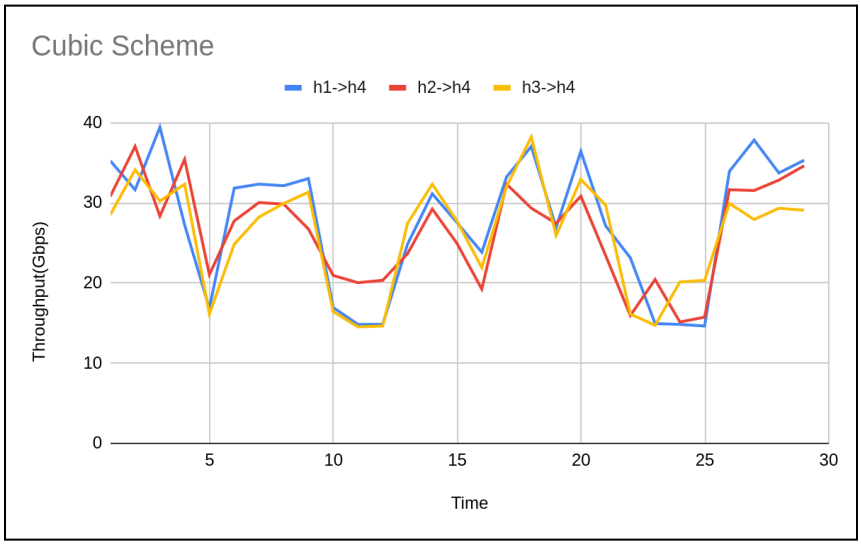


Congestion control mechanism = bbr

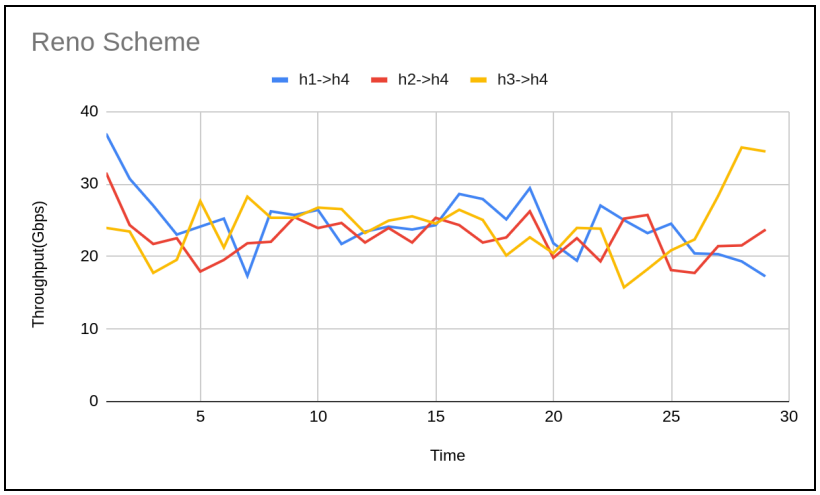
**Command:** sudo mn -c; python3 Q2.py --config=c --congestion=bbr



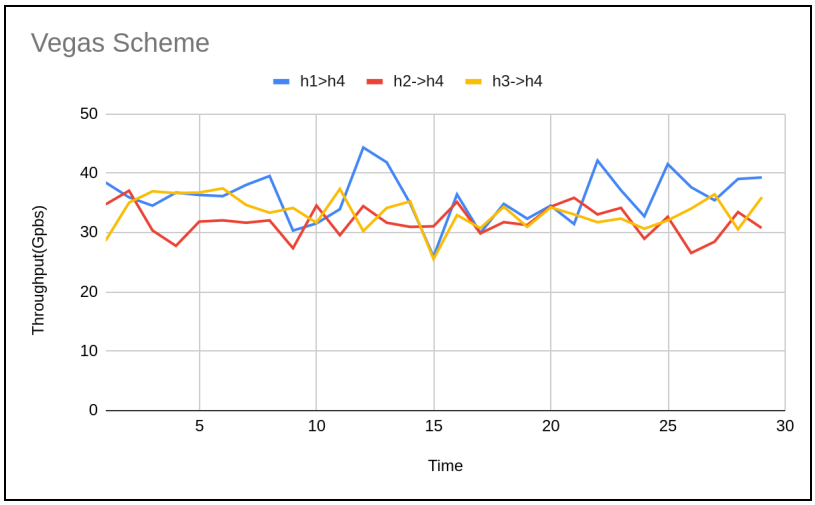

# Graph



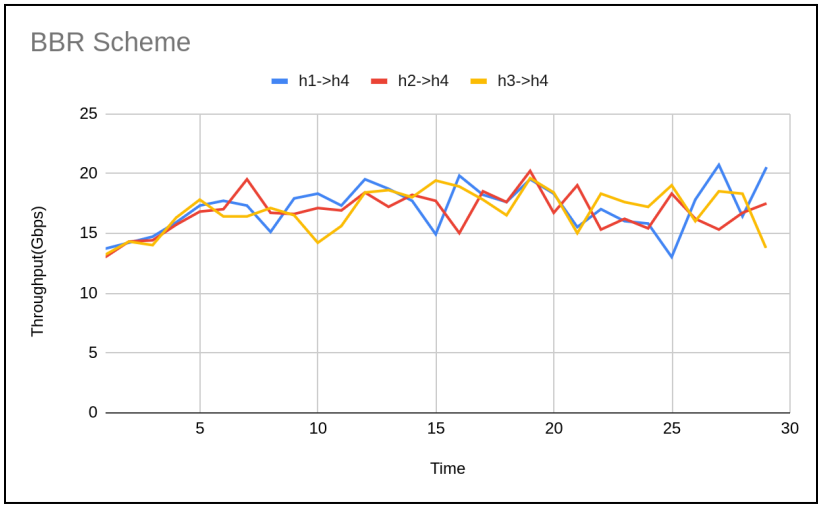

**Avg.Throughput:** 27.5 Gbps




**Avg.Throughput:** 24.1 Gbps




**Avg.Throughput:** 35.6 Gbps




**Avg.Throughput:** 15.7 Gbps

**Observation:** The Avg. Throughput for each scheme has reduced. The reason being that there are multiple connections and each connection competes for the available bandwidth, and as the number of connections increases, the available bandwidth per connection decreases, leading to lower throughput.

**Avg. Throughput Order:** Vegas > Cubic > Reno > Bbr

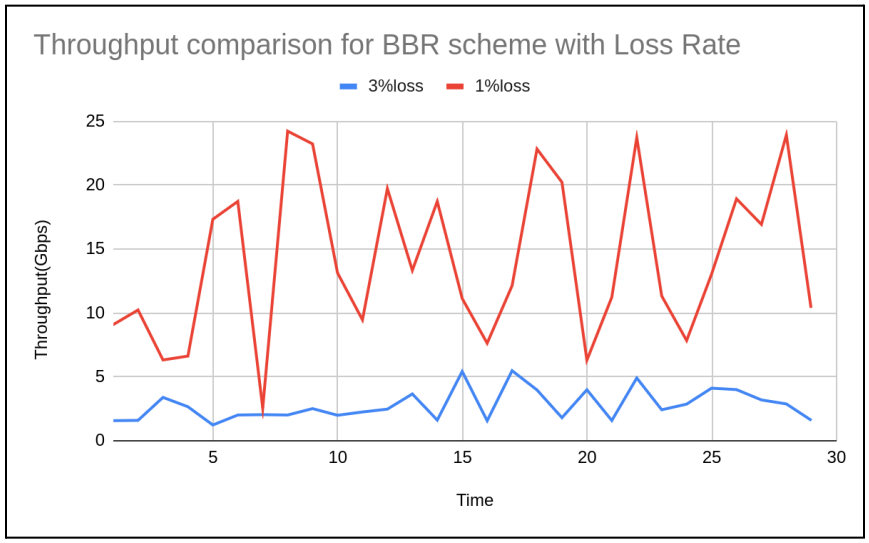
## d. Link Loss Parameter

**Command Used:** sudo mn -c; python3 Q2.py --config=d --congestion={congestion} --loss={1 or 3}

The command on execution generates a txt file containing the Throughput information along with bandwidth.

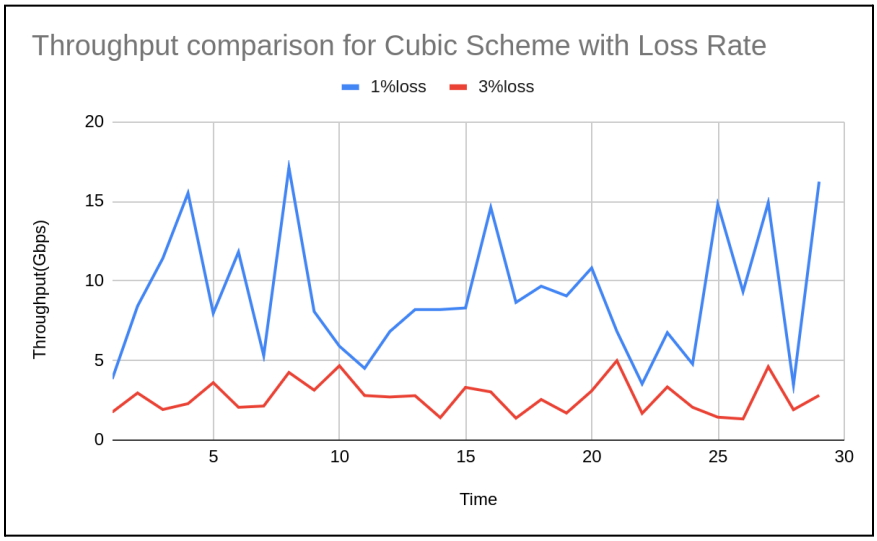
# Graph

**Bbr congestion scheme:**



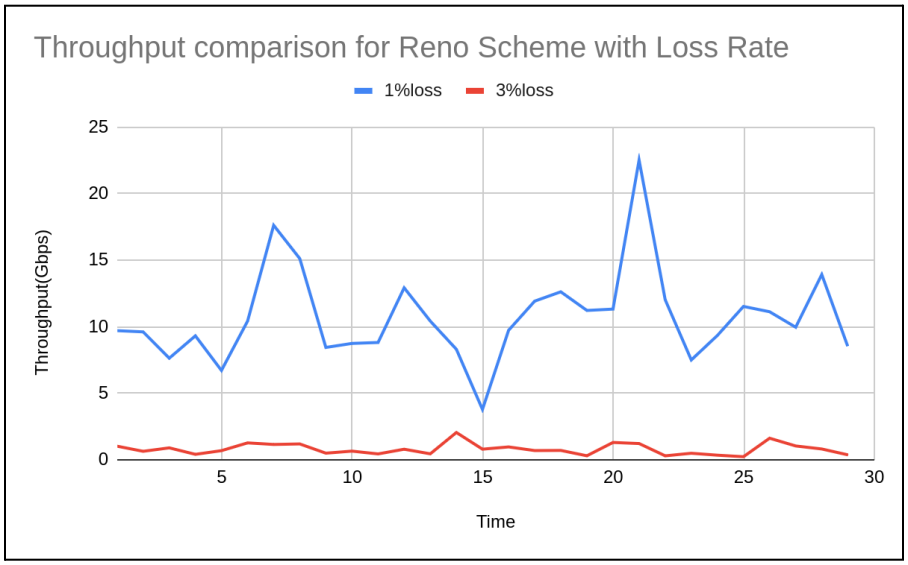

**Avg.Throughput with 1%Loss:** 14.1 Gbps
**Avg.Throughput with 3%Loss:** 2.34 Gbps

**Cubic congestion scheme:**



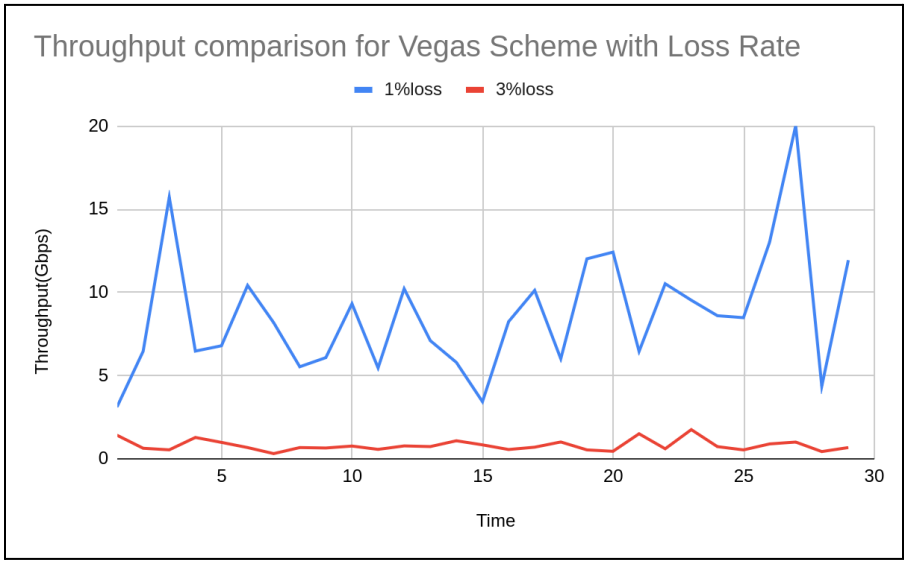

**Avg.Throughput with 1%Loss:** 9.05 Gbps
**Avg.Throughput with 3%Loss:** 2.41 Gbps

**Reno congestion scheme:**




**Avg.Throughput with 1%Loss:** 10.8 Gbps
**Avg.Throughput with 3%Loss:** 0.776 Gbps

**Vegas congestion scheme:**




**Avg.Throughput with 1%Loss:** 8.58 Gbps
**Avg.Throughput with 3%Loss:** 0.765 Gbps

**Observation:** The throughput gets reduced to ⅓ when there is 1% loss. Also, there is a significant amount of throughput reduction as the loss went from 1% to 3%, around 1o times reduction.