University of Dhaka

Department of Computer Science and Engineering

CSE-3111 : Computer Networking Lab

Lab Report 6: Implementation of TCP Reno and New Reno congestion control algorithms and their performance analysis.

Submitted By:

Name : Abdullah Al Mahmud

Roll No : 15

Name : Zisan Mahmud

Roll No : 23

Submitted On:

March 16, 2024

Submitted To:

Dr. Md. Abdur Razzaque

Dr. Md Mamunur Rashid

Dr. Muhammad Ibrahim

Mr. Md. Redwan Ahmed Rizvee

1 Introduction

Congestion control is a mechanism in transport layer that controls the entry of data packets in the network, enabling a better use of a shared network infrastructure and avoiding congestive collapse. *TCP Reno* is a well implemented TCP congestion control algorithm to detect network congestion and adjust data transmission rates accordingly. *TCP New Reno* is the extension of TCP Reno. It overcomes the limitations of Reno.

In this lab our main goal is to implement *TCP Reno* and *TCP New Reno* congestion control algorithm and study their performance under different network condition. The result of this lab will provide insights into the behaviour of *TCP Reno* and *TCP new Reno* under different network condition and help us understanding the effectiveness of network congestion control algorithm.

2 Objectives

The preliminary objective of this lab is to,

- understand and implement the TCP Reno and New Reno congestion control algorithms
- compare their performance under different network condition
- analyze the effectiveness of congestion control algorithm

3 Theory

TCP Reno is a protocol that plays a crucial role in computer networking. It is responsible for effectively managing data transmission between different devices on the internet, ensuring that data packets are delivered efficiently and without errors. Without such protocols, the internet as we know it today would not exist.
The basic principle behind TCP Reno is to adjust the transmission rate based on how quickly packets are being acknowledged by the receiver. If packets are being acknowledged quickly, it means that there is enough available bandwidth to send more data. If packets are not being acknowledged quickly enough, it means that some of the packets may have been lost or

delayed due to network congestion, so TCP slows down its transmission rate to avoid further congestion.
TCP Reno is an extension of the earlier TCP Tahoe algorithm, and it introduces a new mechanism called *fast recovery* to improve network performance. TCP Reno follows four steps for congestion control:

- **Slow Start:** When a TCP connection begins, the value of **cwnd** is typically initialized to a small value of 1 MSS. The available bandwidth of the TCP sender may be much larger than 1 MSS/RTT, the TCP sender would like to find the amount of available bandwidth quickly. Thus, in the slow-start state, the value of cwnd begins at 1 MSS and increases by 1 MSS every time a transmitted segment is first acknowledged, received, doubling the window size every round trip time (RTT) until the slow start threshold is reached.

- **Congestion Avoidance:** Once the slow start threshold is reached, the congestion window size is increased linearly, by 1 /cwnd for each ACK received, where cwnd is the current congestion window size.

- **Fast Re-transmit:** When three duplicate ACKs are received, TCP Reno assumes that a packet has been lost and immediately re-transmits the lost packet.

- **Fast Recovery:** After re-transmitting the lost packet, TCP Reno enters the fast recovery phase. The congestion window size is halved and then kept constant until all lost packets are re-transmitted. After that, the congestion avoidance phase is entered, and the congestion window size is increased linearly.

TCP Tahoe and TCP Reno are similar in many ways, but there are some key differences between them.

- **Fast Recovery:** The most significant difference between TCP Tahoe and TCP Reno is the way they handle packet loss. In TCP Tahoe, when a packet loss is detected, the congestion window is reduced to 1 and the slow start phase is entered. This means that the congestion window size is increased exponentially until the slow start threshold is reached, and then increased linearly.

  In TCP Reno, a fast recovery phase is added. When a packet loss is detected using 3 duplicate ACKs, the congestion window is halved, 3 is added to it, and the fast recovery phase is entered. In this phase,

the congestion window size is kept constant until all lost packets are retransmitted. After that, the congestion avoidance phase is entered, and the congestion window size is increased linearly.

- **Congestion Window Size:** In TCP Tahoe, the congestion window size is reduced to 1 when a packet loss is detected. This can lead to a significant decrease in throughput, especially in high-bandwidth networks. In TCP Reno, the congestion window size is halved when a packet loss is detected. This means that the throughput is not reduced as much as in TCP Tahoe, and the network can recover more quickly from congestion. In summary, the main difference between TCP Tahoe and TCP Reno is the way they handle packet loss. Figure 3.52 illustrates the evolution of TCP's congestion window for both TCP Tahoe and Reno.

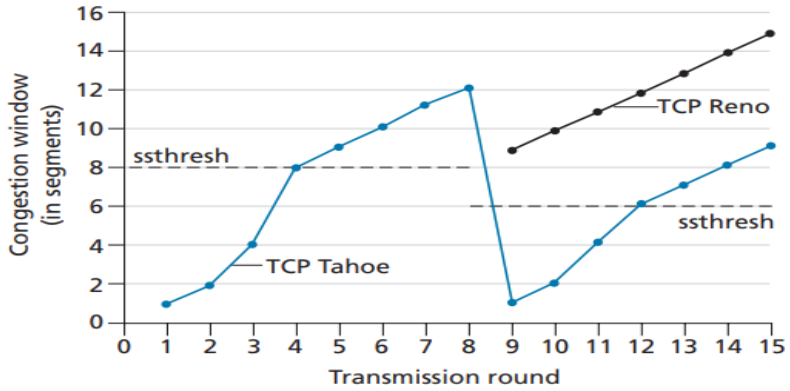

Figure 1: Evolution of TCP's congestion window (Tahoe and Reno)

In this figure, the threshold is initially equal to 8 MSS. For the first eight transmission rounds, Tahoe and Reno take identical actions. The congestion window climbs exponentially fast during slow start and hits the threshold at the fourth round of transmission. The congestion window then climbs linearly until a triple duplicate- ACK event occurs, just after transmission round 8. Note that the congestion window is 12 • MSS when this loss event occurs. The value of ssthresh is then set to 0.5 • cwnd = 6 • MSS. Under TCP Reno, the congestion window is set to cwnd = 9 • MSS and then grows linearly. Under TCP Tahoe, the congestion window is set to 1 MSS and grows exponentially until it reaches the value of ssthresh, at which point

it grows linearly.

TCP New Reno:
TCP New Reno is the extension of TCP Reno. It overcomes the limitations of Reno. TCP Reno is the second variant of the TCP which came up with an in-built congestion algorithm. Congestion handling was not an integral part of the original TCP/IP suite. TCP Reno is the extension of TCP Tahoe, and NewReno is the extension of TCP Reno. In Reno, when packet loss occurs, the sender reduces the cwnd by 50% along with the ssthresh value. This would allow the network to come out of the congestion state easily. But Reno suffered from a very critical backlog which hurts its performance.

When multiple packets are dropped in the same congestion window (say 1-10) then every time it knows about a packet loss it reduces the cwnd by 50%. So, for 2 packet loss, it will reduce the cwnd by 4 times (50% twice). But, one reduction of 50% per congestion window was enough for recovering all those lost packets. Say cwnd=1024 and 10 packets are dropped in this window, Reno will reduce cwnd by 50% 10 times, finally cwnd=1024/210= 1, it is astonishing. It would take 10 RTTs by the sender to again grow its cwnd up to 1024 using slow start leave alone the AIMD algorithm.

Limitation of Reno:

1. It takes a lot of time to detect multiple packet losses in the same congestion window.
2. It reduces the congestion window multiple times for multiple packet loss in the same window, where one reduction was sufficient.

4 Methodology

TCP Reno

1. **Slow start:** First we implement slow start, that means we increase the congestion window size exponentially until the threshold is reached.
2. **Congestion avoidance:** When cwnd size reaches the ssthres, we stop the exponential increase of cwnd size and increase the congestion window size linearly.

3. **Fast retransmit:** Then retransmit lost packets immediately after receiving three duplicate ACKs.

4. **Fast recovery:** Keep the congestion window size constant after retransmitting lost packets.

TCP New Reno

1. Same as TCP Reno.

2. Define a time count equal to half window size.

3. When receive an acknowledgement, start a timer. For three duplicate acknowledgement if the timer is less than time count then do not go to faste recovery, else go to fast recovery.

Analyze Results

1. We collect performance metrics throughput, packet loss rate, and round-trip time (RTT).

2. Finally, we compare the results under different network conditions between TCP Tahoe and TCP Reno.

5 Experimental result

5.1 TCP Reno Sender

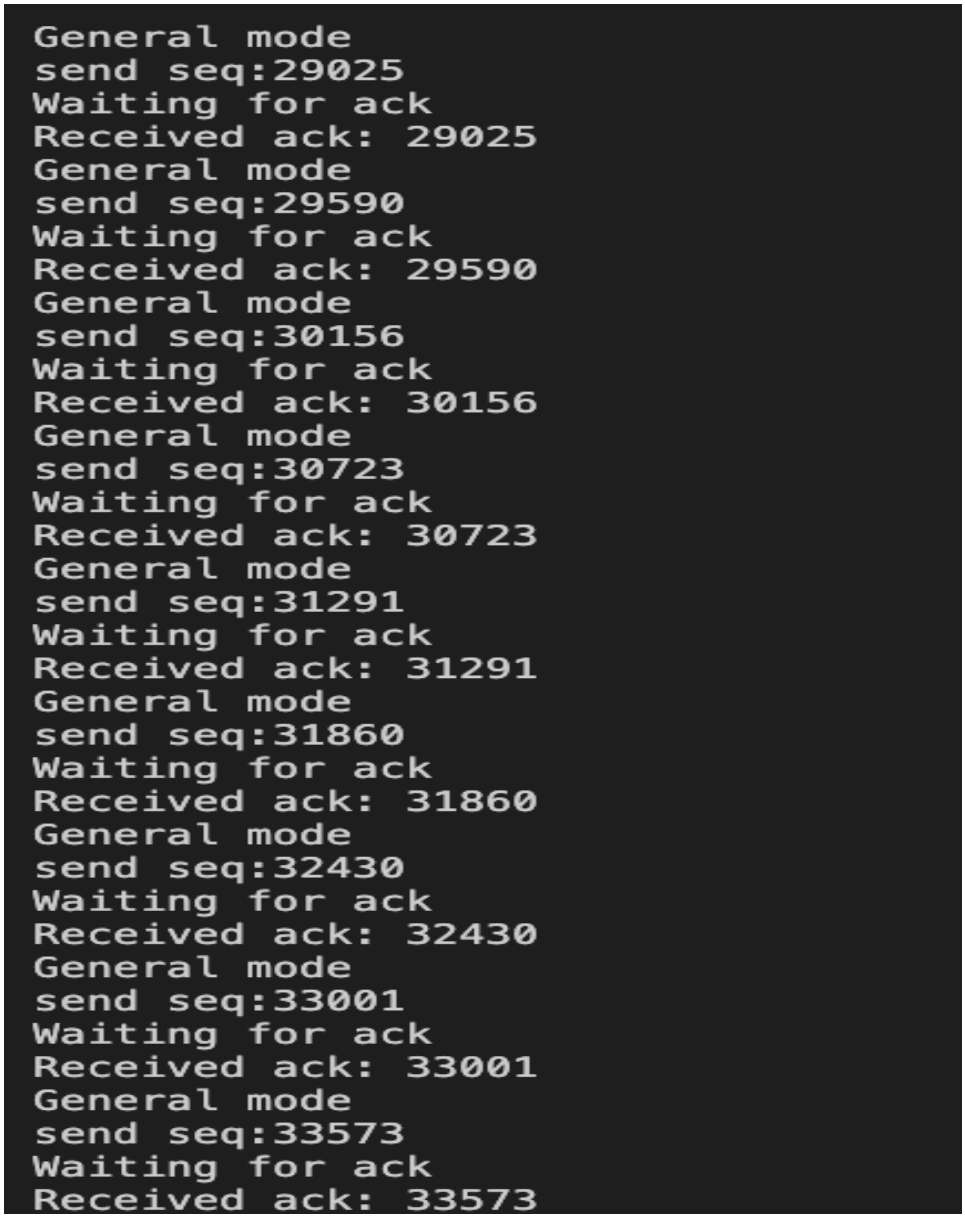

Figure 2: TCP Reno sender

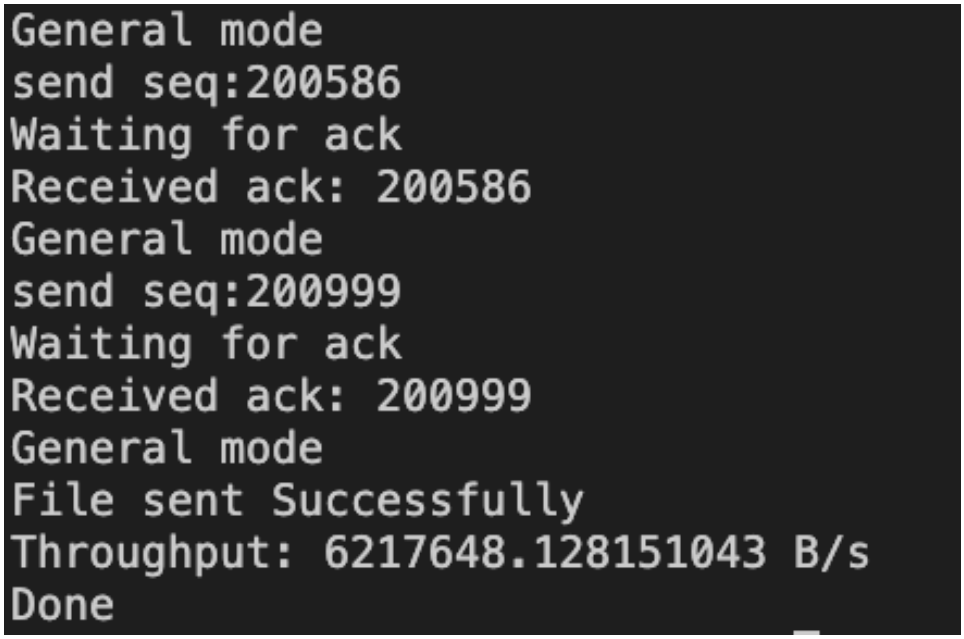

Figure 3: TCP Reno sender with throughput

5.2 TCP Reno receiver

```
Connected to server
Sent packet with seq 0
Received packet with seq 1
Received 1 and wrote to file
Received packet with seq 3
Received 3 and wrote to file
Received packet with seq 7
Received 7 and wrote to file
Received packet with seq 15
Received 15 and wrote to file
Received packet with seq 31
Received 31 and wrote to file
Received packet with seq 63
Received 63 and wrote to file
Received packet with seq 127
Received 127 and wrote to file
Received packet with seq 255
Received 255 and wrote to file
Received packet with seq 511
Received 511 and wrote to file
Received packet with seq 1023
Received 1023 and wrote to file
Received packet with seq 1536
Received 1536 and wrote to file
Received packet with seq 2050
Received 2050 and wrote to file
Received packet with seq 2565
Received 2565 and wrote to file
Received packet with seq 3081
Received 3081 and wrote to file
Received packet with seq 3598
Received 3598 and wrote to file
Received packet with seq 4116
Received 4116 and wrote to file
Received packet with seq 4635
Received 4635 and wrote to file
Received packet with seq 5155
Received 5155 and wrote to file
Received packet with seq 5676
Received 5676 and wrote to file
Received packet with seq 6198
Received 6198 and wrote to file
Received packet with seq 6721
Received 6721 and wrote to file
Received packet with seq 7245
Received 7245 and wrote to file
Received packet with seq 7770
Received 7770 and wrote to file
Received packet with seq 8296
Received 8296 and wrote to file
Received packet with seq 8823
Received 8823 and wrote to file
Received packet with seq 9351
Received 9351 and wrote to file
Received packet with seq 9880
Received 9880 and wrote to file
```

Figure 4: TCP Reno Receiver

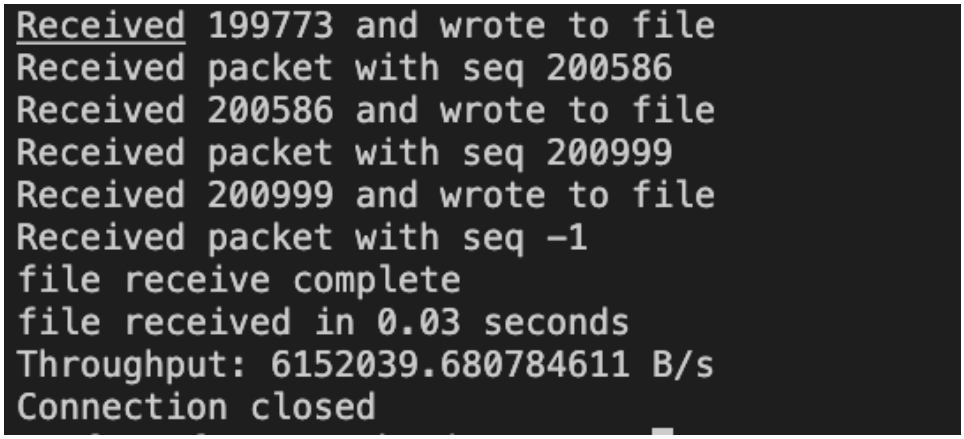

Figure 5: TCP Reno receiver with throughput

5.3 TCP New Reno Sender

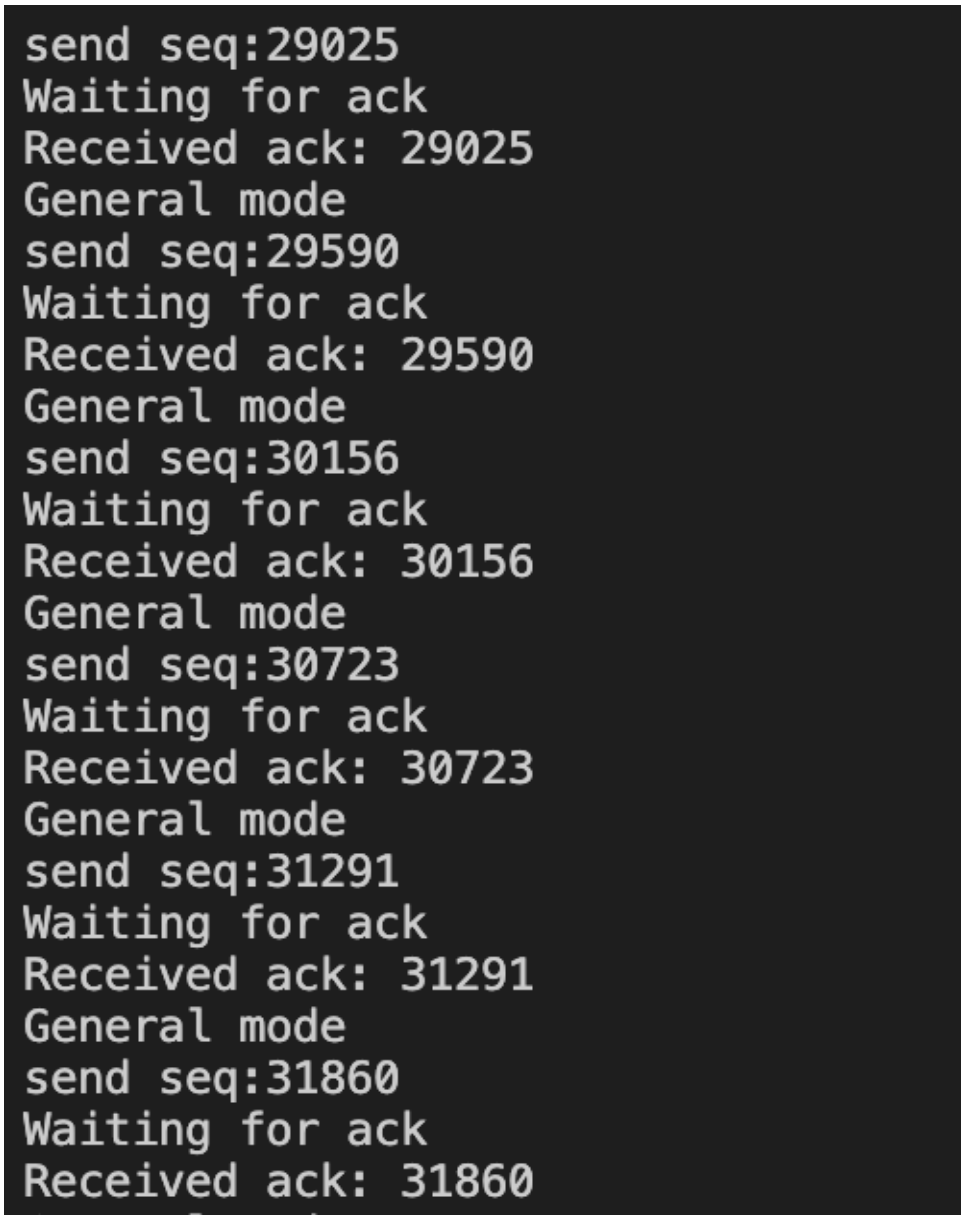

Figure 6: TCP New Reno sender

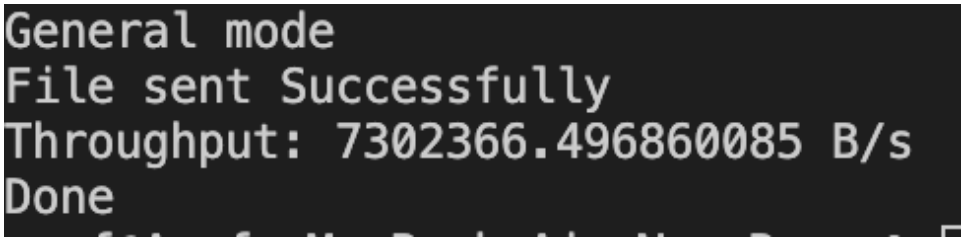

Figure 7: TCP New Reno sender with throughput

5.4 TCP New Reno receiver

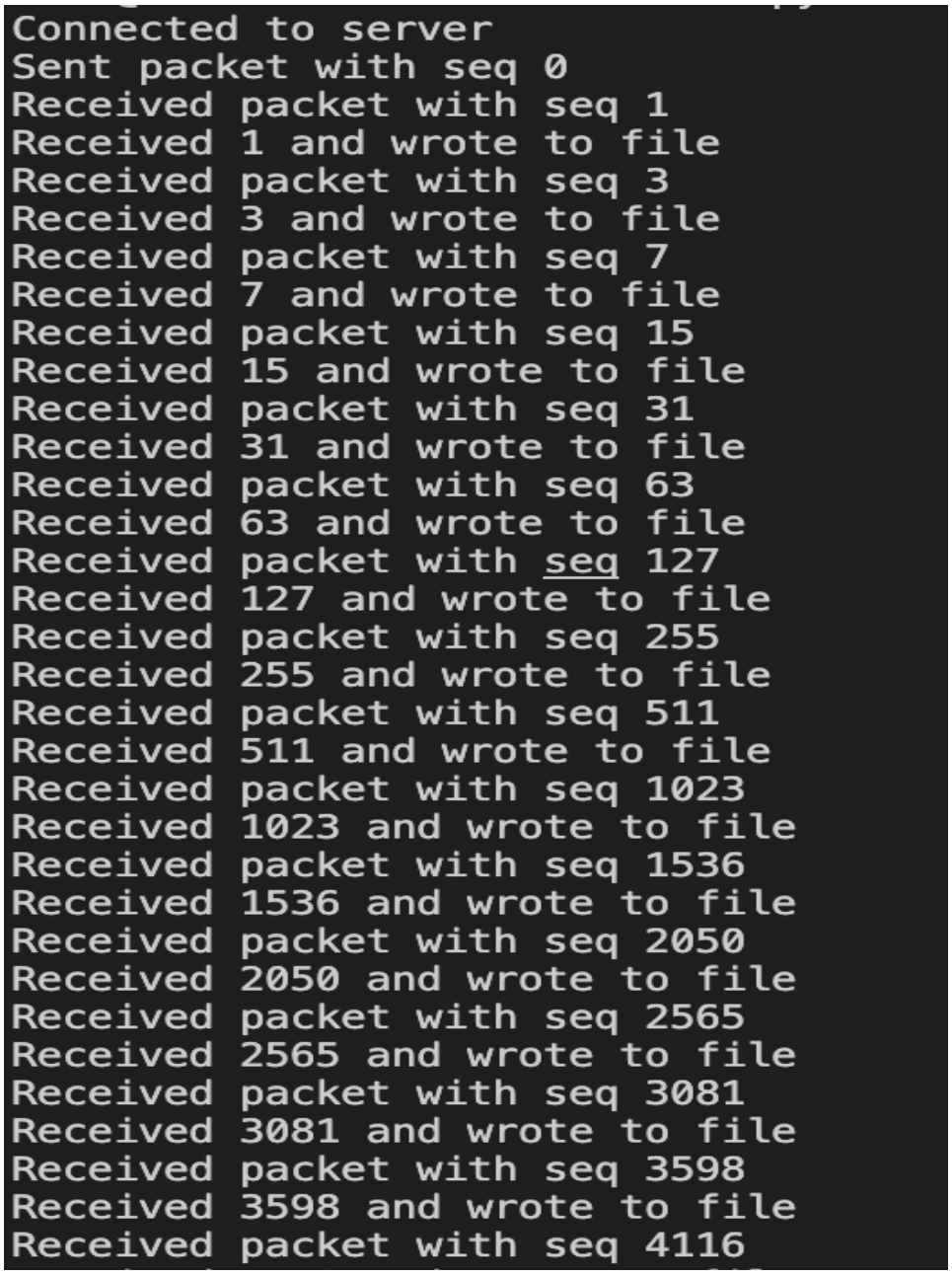

Figure 8: TCP New Reno Receiver

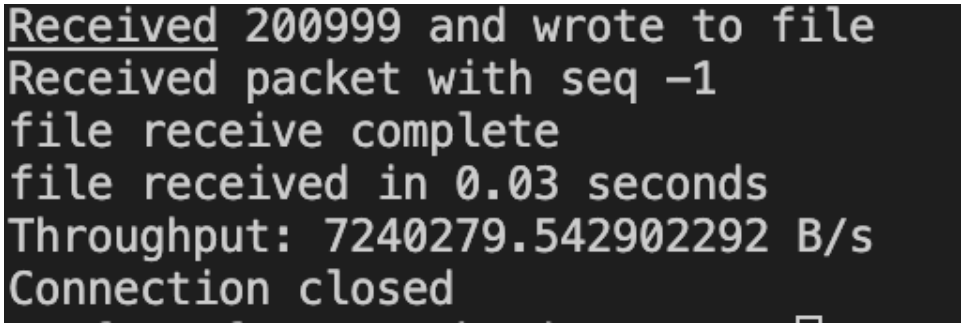

Figure 9: TCP New Reno receiver with throughput

5.5 Result Analysis

Throughput comparison between TCP Reno and TCP New Reno

TCP New Reno does not simple half the congestion window size for each packet loss, rather it will check if the acknowledgement of the retransmitted packet is new or not in the sense that all the packets of that particular cwnd are ACKed by the receiver or not. Thus the throughput increses in case of TCP New Reno compared with TCP Reno.

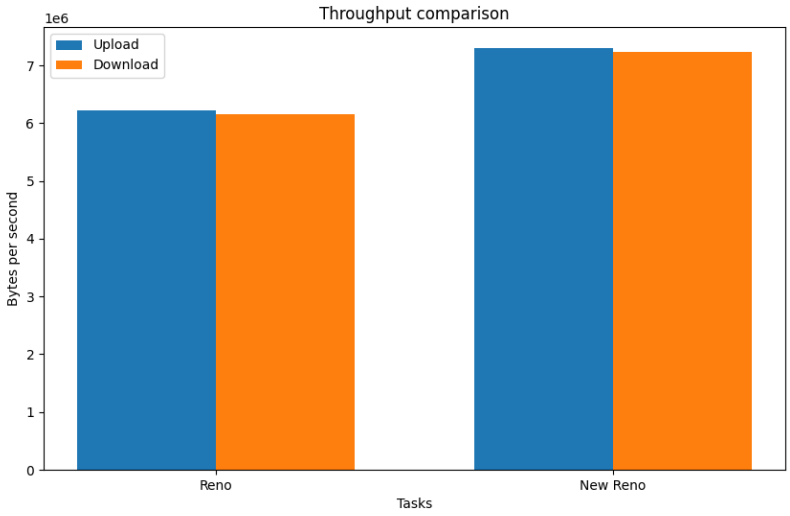

Figure 10: TCP Reno vs TCP New Reno throughput

Change of congestion window size with time

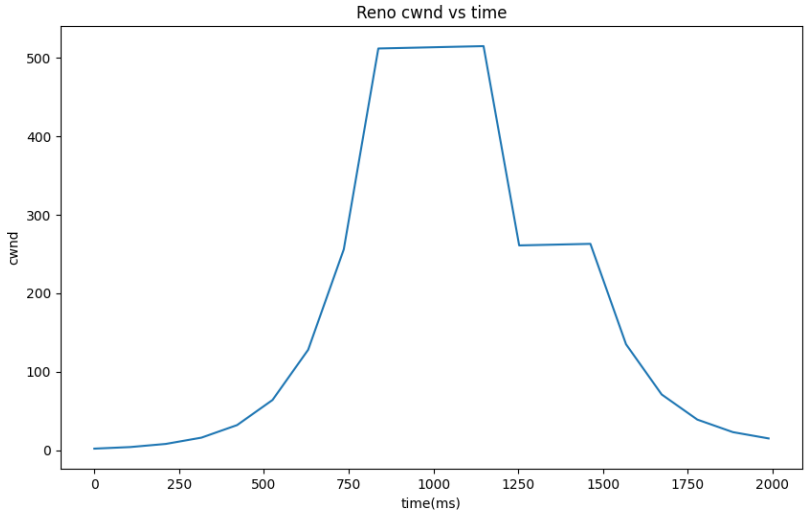

Figure 11: TCP Reno congestion window vs time

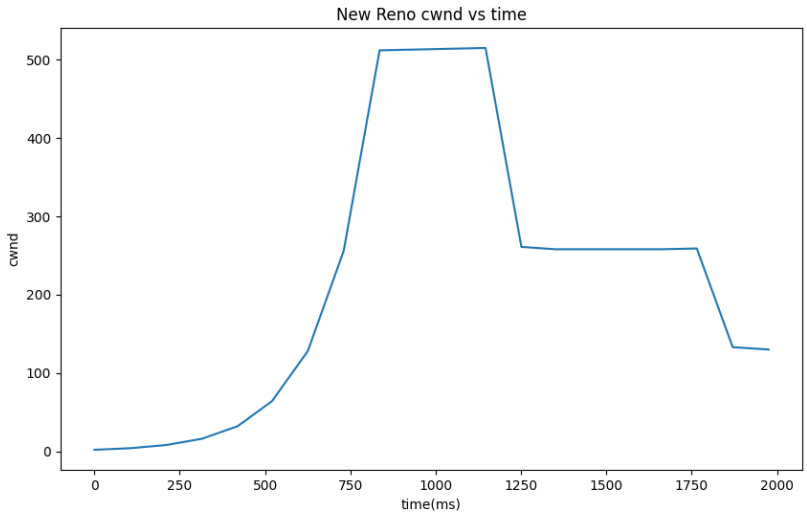

Figure 12: TCP New Reno congestion window vs time

Packet loss rate

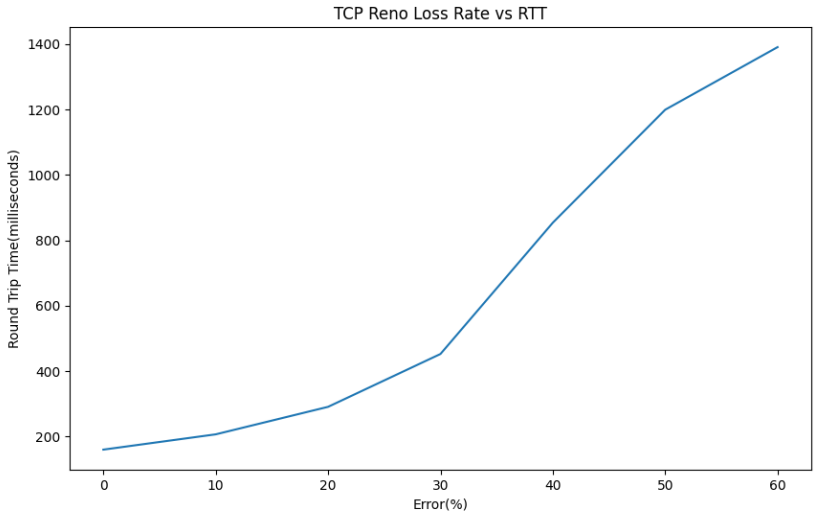

Figure 13: TCP Reno packet loss rate

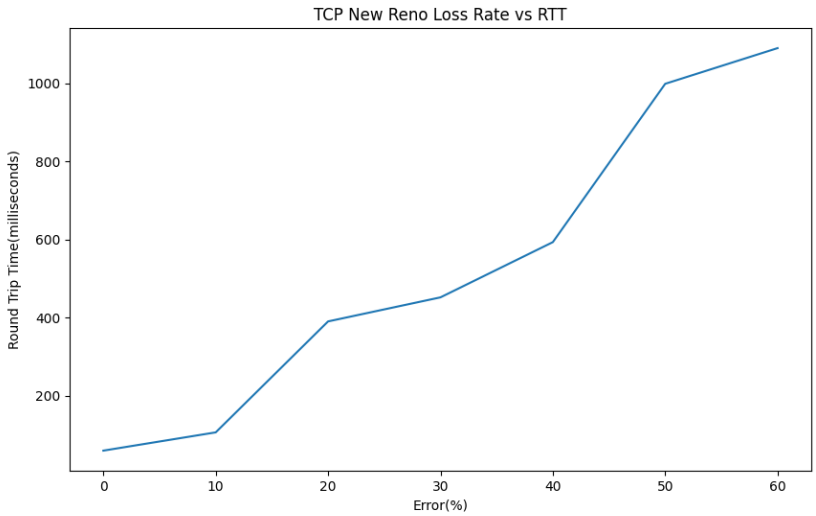

Figure 14: TCP New Reno packet loss rate

From the aforementioned graph, we can see that TCP Reno has a greater packet loss rate compared with TCP new Reno. This is because TCP New Reno can handle multiple packet loss.

References

[1] James F. Kurose and Keith W. Ross, *Computer Networking: A Top-Down Approach*, 8th Edition, Pearson,, 2022.

[2] GeeksforGeeks, *TCP Tahoe and TCP Reno*, `https://www.geeksforgeeks.org/tcp-tahoe-and-tcp-reno/`

[3] Tutorials Point, *TCP Reno with Example*, `https://www.tutorialspoint.com/tcp-reno-with-example`

[4] GeeksforGeeks, *TCP Reno with example*, `https://www.geeksforgeeks.org/tcp-reno-with-example/`