University of Dhaka

Department of Computer Science and Engineering

CSE-3111 : Computer Networking Lab

Lab Report 5: Implementation of TCP Reno and New Reno congestion control algorithms and their performance analysis.

Submitted By :

Name: Md. Emon Khan

Roll No : 30

Name: Mahmudul Hasan

Roll No : 60

Submitted To :

Dr. Md. Abdur Razzaque

Dr. Md. Mamun Or Rashid

Dr. Muhammad Ibrahim

Md. Redwan Ahmed Rizvee

Submitted On : March 16, 2024

Contents

1 Introduction

The primary goal of Lab-5 is implement the TCP reno and new Reno congestion control algorithm. The experiment focuses on two key. At first we implemeted the TCP Reno congestion control alogrithm and then another server for TCP new reno congestion control algorithm. TCP new reno introduces the concept of fast recovery to enhance network efficiency by responding to packet loss events. Through this experiment, we aim to fully understand the workings of TCP Reno and new Reno, including its phases of slow start, congestion avoidance, fast retransmit, and fast recovery.

1.1 Objectives

Some of the preliminary objectives of this lab experiment are:

- To understand the TCP Reno and New Reno congestion control algorithms.
- Compare TCP Reno and New Reno performances.

2 Theory

2.1 TCP Reno: Congestion Control Algorithm

TCP Reno is a congestion control algorithm used within the Transmission Control Protocol (TCP), a core protocol of the internet protocol suite. It is an extension of the earlier TCP Tahoe algorithm, and it introduces a new mechanism called "fast recovery" to improve network performance. The operation of TCP Reno can be broken down into several phases:

2.1.1 Slow Start

When a TCP connection is established or reestablished after a period of inactivity, TCP Reno begins in the slow start phase. During slow start, the sender increases the congestion window size exponentially. Initially, the congestion window size is set to one segment (usually one packet). For each acknowledgment (ACK) received from the receiver, indicating successful receipt of a packet, the congestion window size is incremented by one segment. This results in exponential growth of the congestion window size.

2.1.2 Congestion Avoidance

Once the congestion window size reaches a certain threshold, known as the slow start threshold (ssthresh), TCP Reno transitions to the congestion avoidance phase. In congestion avoidance, the congestion window size increases linearly rather than exponentially. For every ACK received, the congestion window size is incremented by 1/cwnd, where cwnd represents the current congestion window size.

2.1.3 Fast Retransmit

In the event of packet loss, TCP Reno employs a mechanism called fast retransmit to quickly recover from the loss without waiting for a timeout. When the sender receives three duplicate ACKs for the same sequence number, it confirm that a packet has been lost. Instead of waiting for a timeout, TCP Reno immediately retransmits the missing packet.

2.1.4 Fast Recovery

Upon detecting packet loss and initiating fast retransmit, TCP Reno enters the fast recovery phase. In fast recovery, the congestion window size is reduced by half and then maintained at that value until all outstanding lost packets are retransmitted. After retransmitting the lost packets, TCP Reno exits the fast recovery phase and enters congestion avoidance, where the congestion window size resumes increasing linearly.

2.2 TCP new Reno

TCP New Reno extends the capabilities of TCP Reno, addressing some of its limitations. In TCP Reno, when packet loss occurs, the sender reduces the congestion window size (cwnd) by 50% along with the slow start threshold (ssthresh) value. While this approach aims to facilitate quick recovery from congestion, it can result in inefficiencies, especially when multiple packets are dropped within the same congestion window. For instance, if 10 packets are lost within a single congestion window, TCP Reno would reduce the congestion window size by 50% for each loss, potentially leading to a significant decrease in throughput and a backlog in recovery. TCP New Reno addresses these challenges by refining the congestion control mechanisms. Instead of reducing the congestion window size for each packet loss within the same congestion window, TCP New Reno introduces the concept of partial acknowledgments. This approach allows the sender to better discern the

extent of packet loss within a congestion window and adjust its congestion control strategies accordingly. In TCP New Reno, when packet loss occurs, the sender estimates partial acknowledgments to determine the extent of packet loss within the current congestion window. If multiple packets are lost within the same congestion window, Rather than indiscriminately reducing the congestion window size for each loss, TCP New Reno adjusts its behavior based on the acknowledgment pattern.

3 Methodology

1. **Initialization**:
    - The server initializes a socket using TCP/IP and binds it to a specific address and port.
    - The server listens for incoming connections from clients.
    - Upon connection, the client and server establish a TCP connection.
2. **Packet Structure**:
    - Data packets consist of a header and payload, where the header contains fields such as sequence number, acknowledgment number, acknowledgment flag, checksum, etc.
    - The payload contains the actual data being transmitted.
3. **Congestion Control**:
    - The server implements congestion control mechanisms to regulate data flow, including slow start, congestion avoidance, and fast recovery.
    - The congestion window size is adjusted dynamically based on network conditions.
4. **Timeout Handling**:
    - Both client and server utilize timeout mechanisms to handle packet loss and ensure reliable data transmission.
    - If a packet is not acknowledged within a specified timeout period, it is retransmitted.
5. **Error Detection**:

- Checksums are calculated for each packet to detect errors during transmission.
- The receiving end verifies the checksum to ensure data integrity.

6. **Flow Control**:
   - Flow control mechanisms are incorporated to prevent the sender from overwhelming the receiver.
   - Both sender and receiver maintain a receive window size (RWND) to regulate the amount of data sent.

7. **Close Connection**:
   - Upon completion of data transmission, the client and server close the connection gracefully.

4 Experimental result

Some Snapshots of the terminal output for each of these tools.

4.1 Tcp Tahoe vs Reno

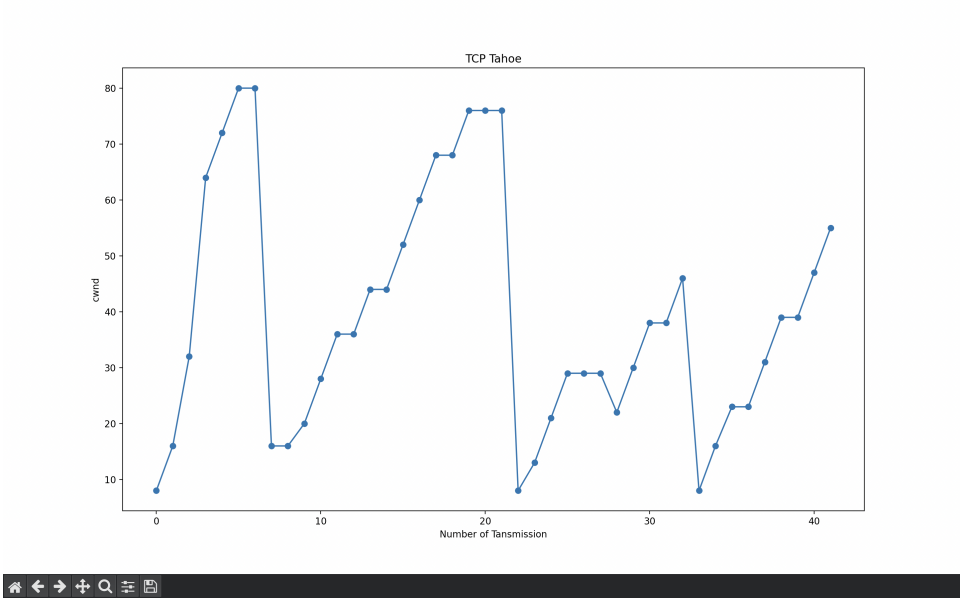

Figure 1: Tcp Tahoe cwnd vs Number of packet sent

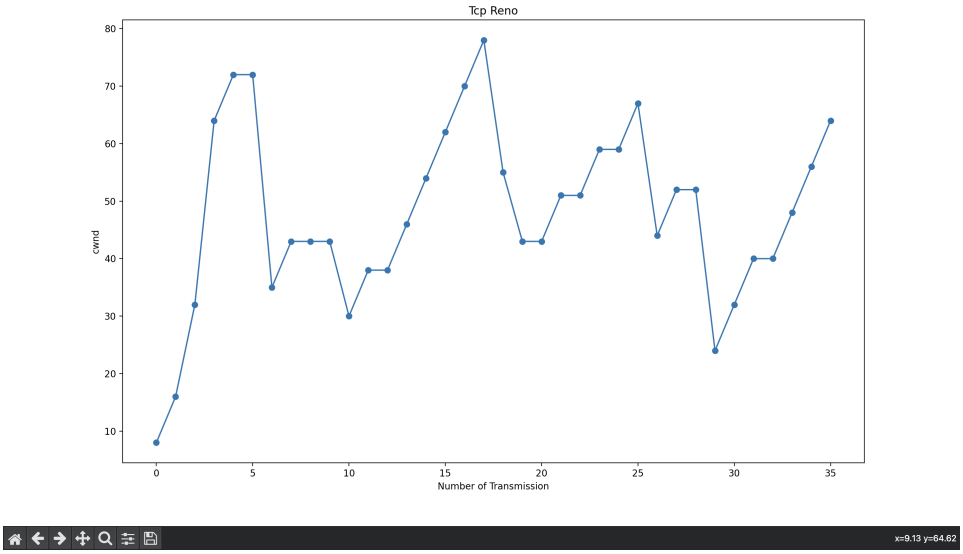

Figure 2: Tcp Reno cwnd vs Number of packet sent

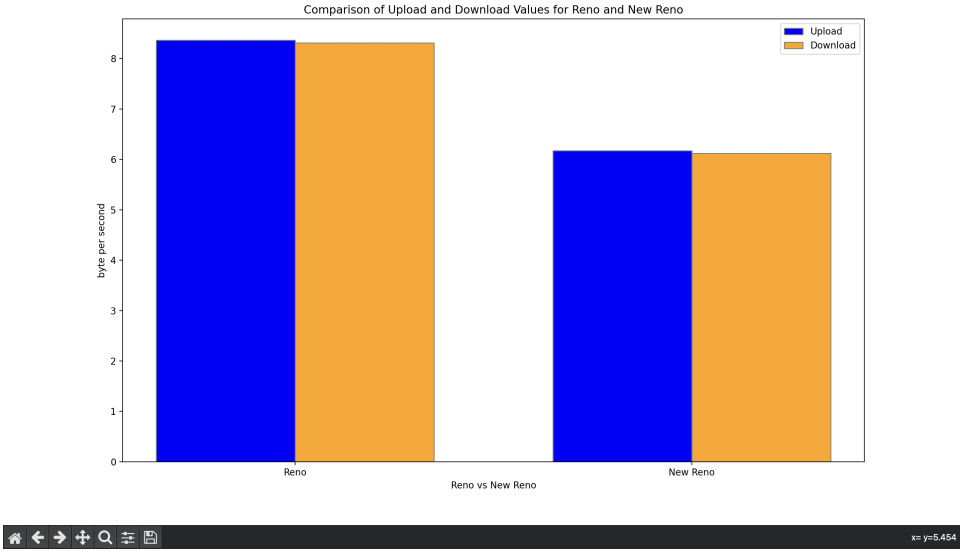

Figure 3: Reno Vs New Reno Throughput Comparisons.(We used different server to measure output for Reno and New Reno

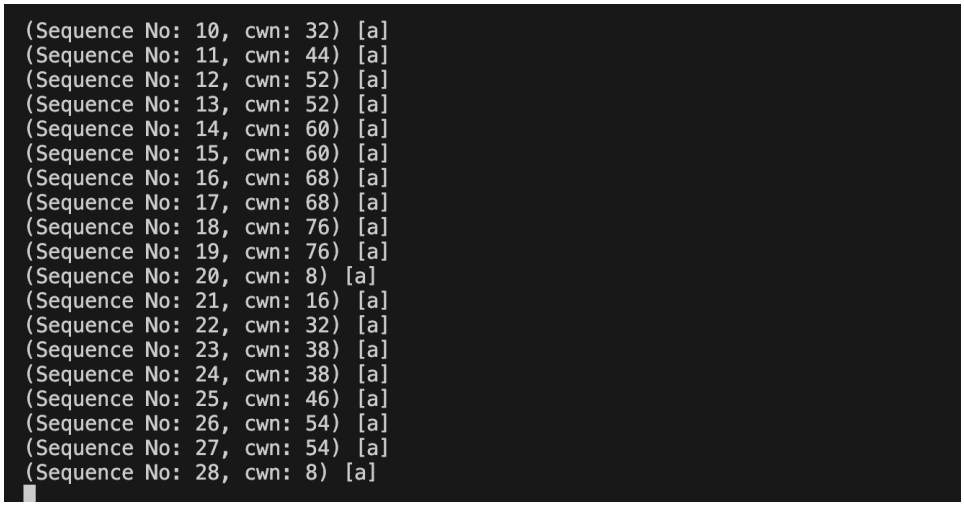

Figure 4: Terminal Output Sequence number and cwnd

5 Experience

- Learned the concept of TCP congestion control
- Understand TCP Reno and new Reno reno congestion algorithm and their impact on network performance.
- Deeper understanding of the trade-offs between congestion window adjustments, packet retransmissions, and network throughput
- Clear understanding of TCP Reno's phases, including slow start, congestion avoidance, fast retransmit, and fast recovery

References

[1] https://www.geeksforgeeks.org/tcp-tahoe-and-tcp-reno

[2] https://www.cs.cmu.edu/~prs/15-441-F14/lectures/rec05-congestion.pdf

[3] https://en.wikipedia.org/wiki/TCP_congestion_control#TCP_Tahoe_and_Reno