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: Meherun Farzana

Roll No : 05

Name: Mohd. Jamal Uddin Mallick

Roll No : 07

Submitted On :

March 16, 2024

Submitted To :

Dr. Md. Abdur Razzaque

Dr. Md. Mamun Or Rashid

Dr. Muhammad Ibrahim

Redwan Ahmed Rizvee

1 Introduction

Transmission Control Protocol (TCP) is a critical component of internet communication, ensuring dependable data transmission across networks. Within TCP, congestion control algorithms play a pivotal role in handling data flow during network congestion. Notably, two prominent variants of TCP congestion control are TCP Reno and TCP New Reno.

TCP Reno is a widely used congestion control algorithm within the Transmission Control Protocol (TCP). It operates by employing a window-based approach to regulate the rate of data transmission in the presence of network congestion. The key principles behind TCP Reno include slow start, congestion avoidance, and fast recovery. In practical terms, it gradually increases transmission rates, carefully adjusts them to prevent congestion, and swiftly recovers from packet loss. Researchers and network engineers study TCP Reno's behavior under various network conditions to understand its effectiveness in managing congestion.

The TCP Reno congestion control algorithm, while widely used, has some limitations:

1. Inability to Detect Multiple Packet Losses in a Single Window:
2. TCP Reno struggles to identify multiple packet losses occurring within a single window size.
3. When several packets are lost simultaneously, Reno may not handle them optimally.
4. It can lead to unnecessary timeouts or multiple fast retransmits and window reductions

Due to these limitations, TCP New Reno emerged as an enhancement to TCP Reno in response to its limitations. By refining the fast recovery mechanism, TCP New Reno enables more precise re-transmissions after encountering multiple packet losses within a single window. This enhancement significantly enhances the overall performance and resilience of TCP congestion control, especially in demanding network conditions.

1.1 Objectives

- To implement the TCP Reno and TCP New Reno congestion control algorithms and assess their effectiveness in managing data transmission rates amidst network congestion through its congestion control mechanism.
- To offer a deeper understanding of both the algorithms' behavior and comparing efficacy in measuring parameters like throughput, packet loss, and delay.

2 Theory

TCP Reno, an integral part of the Transmission Control Protocol (TCP), manages network congestion. Originating from its debut in Reno, Nevada, during a 1990 conference, TCP Reno builds upon the groundwork laid by its predecessor, TCP Tahoe. It introduces an innovative mechanism called "fast recovery" to enhance network efficiency. Functioning through four distinct phases – slow start, congestion avoidance, fast retransmit, and fast recovery – TCP

Reno demonstrates the capacity to effectively manage data transmission within dynamic network environments.

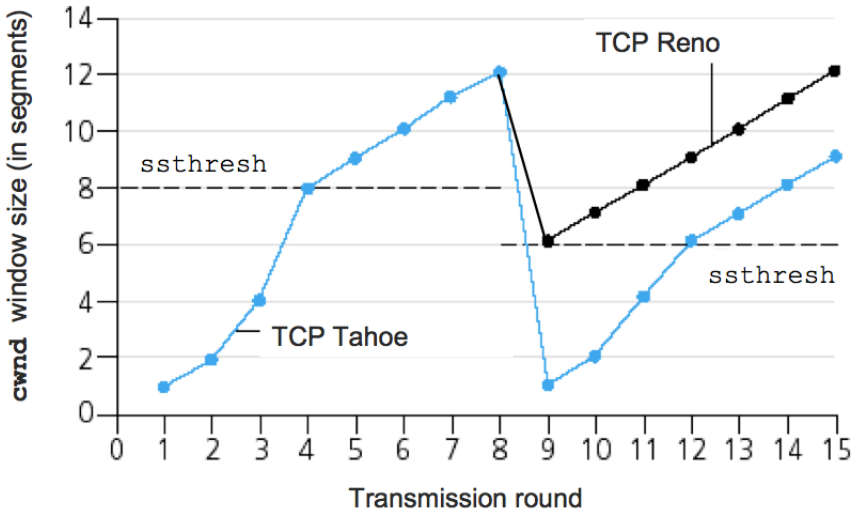

Figure 1: TCP Tahoe and TCP Reno

TCP Reno: A 4 Keys Process

1. **Slow Start:** Starts with a congestion window size of 1. Window size increases by 1 for each received ACK, doubling every round-trip time (RTT). This exponential growth continues until reaching the slow start threshold (ssthresh).

2. **Congestion Avoidance:** After reaching ssthresh, the window size increases linearly. For each received ACK, it grows by 1/current window size (cwnd). This slower growth maintains stability while allowing for gradual window expansion.

3. **Fast Re-transmit:** Triggers upon receiving three duplicate ACKs for the same packet. Interprets this as packet loss and immediately retransmits the missing data.

4. **Fast Recovery:** Initiated after the fast re-transmit. Window size is halved and then increased by 3 units (aggressive adjustment). This adjusted window is maintained until all lost packets are acknowledged. Once complete, the protocol resumes congestion avoidance with linear window growth.

TCP New Reno builds upon TCP Reno. It addresses a limitation in Reno's fast recovery mechanism. In TCP Reno's fast recovery, a single ACK for the retransmitted packet might be misinterpreted as complete recovery, leading to premature window size adjustments and potential performance degradation.

New Reno introduces a more nuanced approach. It utilizes a variable called "recover" to track unacknowledged data. Only when all packets before the "recover" point are acknowledged does New Reno exit fast recovery and adjust the congestion window. This prevents unnecessary reductions and maintains optimal data flow.

TCP New Reno addresses several limitations present in TCP Reno, particularly in how it handles network congestion and packet loss. Here's how TCP New Reno overcomes these limitations:

1. **Handling Partial Acknowledgments:** A sequence of duplicate acknowledgments is seen by TCP Reno as congestion, which causes a quick retransmit and a shortening of the congestion window. This method can be ineffective and unduly cautious, particularly in situations where only a portion of the packets are lost. To combat this, TCP New Reno makes a distinction between duplicate acknowledgments that are the consequence of out-of-order delivery and those that are the result of genuine packet loss. It makes use of this difference to improve its congestion control approach, which enables it to recover from packet loss more smoothly and without cutting the congestion window size in half.

2. **Faster Recovery from Packet Loss:** Timeout-based techniques are a major component of TCP Reno's congestion recovery strategy. TCP Reno may cause extended periods of slower throughput when it resets the congestion window to a minimal value and gradually increases it after a timeout. Faster recovery methods, like rapid retransmit and fast recovery, are introduced by TCP New Reno, allowing it to immediately resume transmission at a rate that is comparable to what it was before the packet loss happened. Compared to TCP Reno, this leads to more effective bandwidth use and lower latency.

3. **Improved Throughput and Utilization:** Compared to TCP Reno, TCP New Reno achieves higher throughput and better bandwidth usage by lessening the effect of packet loss on congestion control and recovery. Its enhanced dynamic adaptation to network conditions enables it to sustain steady data transfer rates even when there is congestion, leading to better overall performance.

4. **Enhanced Responsiveness to Network Conditions:** TCP New Reno is more adaptable to changes in network conditions, such as variations in packet loss rates or available bandwidth, thanks to its finer-grained congestion control techniques and speedier recovery algorithms. In comparison to TCP Reno, TCP New Reno offers superior speed and a more consistent user experience because of its responsiveness, which allows it to adjust to changing network settings more quickly.

Overall, TCP New Reno expands on the core functionality of TCP Reno by adding enhancements and optimizations that increase its resilience, effectiveness, and responsiveness in handling packet loss and network congestion, which eventually improves network performance in contemporary environments.

Table 1: Comparison of TCP Reno and TCP New Reno

Point	**TCP Reno**	**TCP New Reno**
Slow Start	Identical	Identical
Congestion Avoidance	Identical	Identical
Fast Retransmit	Retransmits lost packet upon receiving 3 duplicate ACKs.	Identical to Reno.
Fast Recovery	Reduces cwnd by half, adds 3, keeps constant until all lost packets are acked, then resumes congestion avoidance. May cause premature window adjustments.	Reduces cwnd by half, adds 3, keeps constant until all packets before the "recover" point are acked, then resumes congestion avoidance. "Recover" point tracks unacknowledged data for more accurate recovery assessment.

3 Methodology

1. Build a network of computers (LAN/WAN) for TCP communication.
2. Develop client and server apps using socket programming
3. Implement Sliding window for flow control.
4. Implement 4 phases of TCP Reno: Slow Start (exponential window growth), Congestion Avoidance (linear growth), Fast Retransmit (retransmit on 3 duplicate ACKs), Fast Recovery (adjust window, might be premature in Reno).
5. Define scenarios with varying Bandwidth limitations, network latency (delays), packet loss rates.
6. Assign sender and receiver roles for each scenario.
7. Configure TCP Reno parameters for each scenario.
8. Run data transfer between sender and receiver.
9. Do the same for TCP New Reno except for the part of Fast Recovery.
10. Maintain a "recover" point variable that identifies the location of the potentially lost packet. During fast recovery, wait for acknowledgments (ACKs) for all data before the "recover" point, not just the lost packet itself.

11. Only after receiving these ACKs, signifying a clearer understanding of the network congestion, should the protocol exit fast recovery and resume the linear window growth observed in the congestion avoidance phase.

12. Record: Throughput (data transfer rate), Round-trip time (RTT), Retransmissions, Network utilization.

13. Analyze data to understand and compare the behavior of TCP Reno and New Reno.

4 Experimental Result

The following snapshots were taken after running the server side and client side of TCP Reno and New Reno:

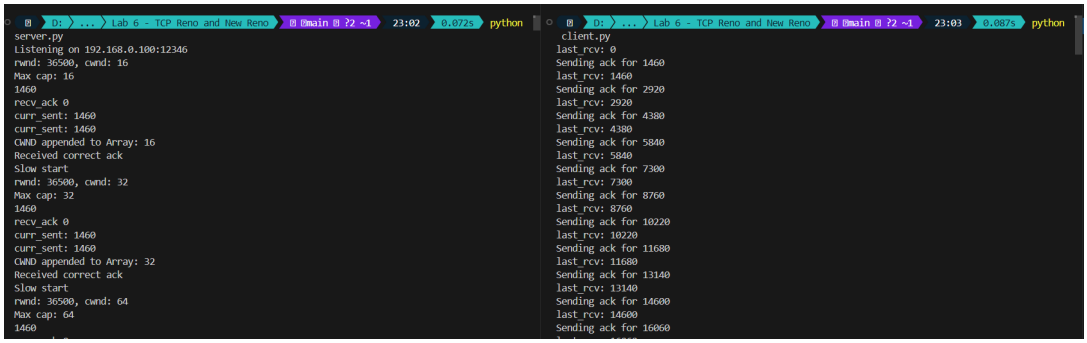

Figure 2: Server and client side of TCP Reno

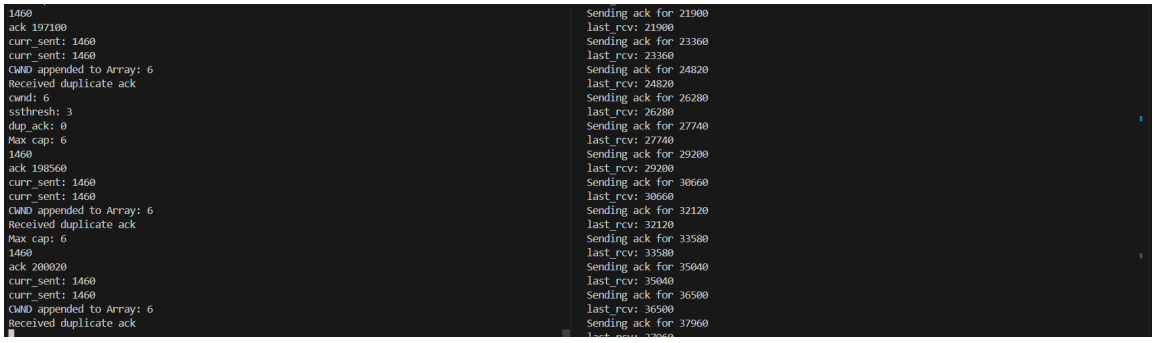

Figure 3: Server and client side of TCP New Reno

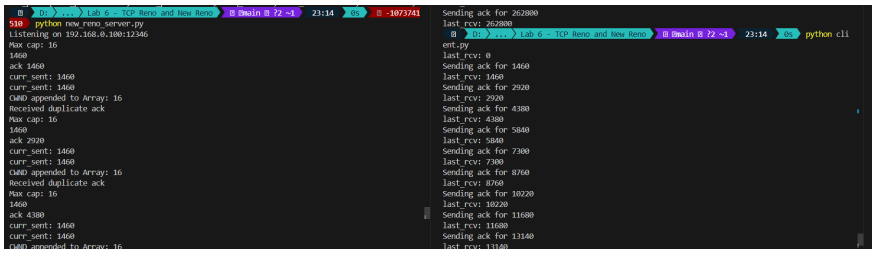

Figure 4: Server and client side of TCP New Reno (2)

The following snapshot was taken after transfer

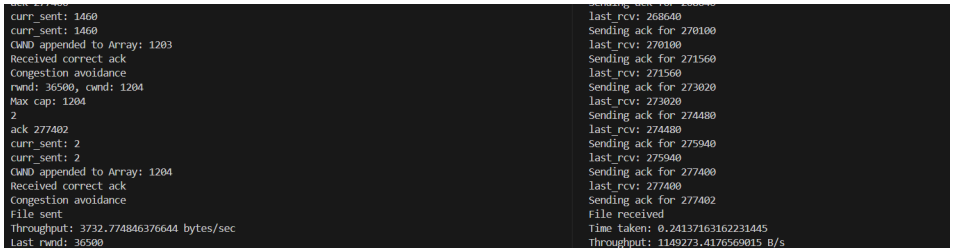

Figure 5: Server and client side of TCP Reno

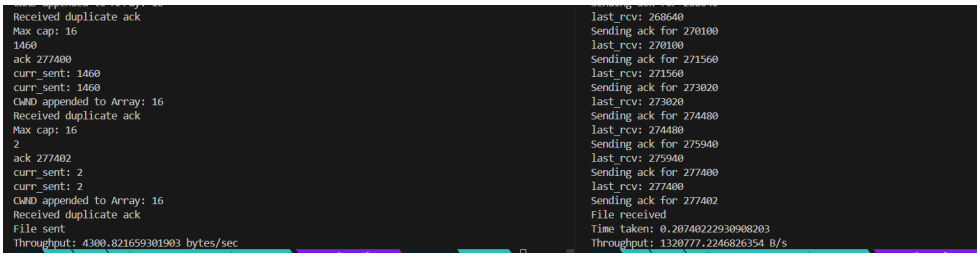

Figure 6: Server and client side of TCP New Reno(3)

5 Result Comparison

Understanding the behavior of TCP congestion control algorithms like Reno and New Reno requires analyzing key metrics throughout data transfer. Here's a breakdown of how these metrics might differ in comparison graphs:

1. **CWND vs. Time**

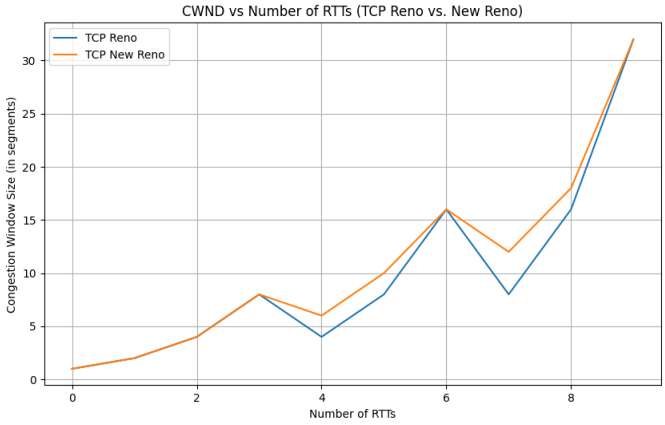

Figure 7: Comparison of CWND vs RTT

TCP Reno: The graph depicting CWND versus Time illustrates the fluctuation of the congestion window throughout the data transmission process in TCP Reno. At the outset, the sender commences with a modest congestion window size, typically matching the maximum segment size (MSS). Upon initiating data transmission, the sender awaits acknowledgments from the receiver. Should these acknowledgments arrive within a specified timeframe, the congestion window size expands, facilitating increased data transmission without inducing network congestion. Conversely, upon detecting packet loss or network congestion, the sender diminishes the congestion window size to avert further congestion and aid network recovery. This graph typically displays a "sawtooth" pattern. During the slow start, the line increases exponentially, representing window size growth. Upon packet loss, a sharp drop signifies window reduction by half. The recovery phase might show a quick rise if the lost packet is acknowledged early, but could exhibit a slower climb if more packets were lost.

TCP New Reno: Similar to Reno in slow start and congestion avoidance, the CWND grows steadily. However, TCP New Reno differs from TCP Reno in its handling of partial acknowledgments during fast recovery. Instead of reducing the congestion window size to the initial value upon receiving duplicate acknowledgments, TCP New Reno adapts by reducing the congestion window size by a smaller amount, enhancing its responsiveness to network conditions and improving throughput. Furthermore, TCP New Reno employs a timeout mechanism to detect packet loss, triggering a reduction in the congestion window size to mitigate congestion and facilitate network recovery.

2. **RTT vs. Error Rate (%)**

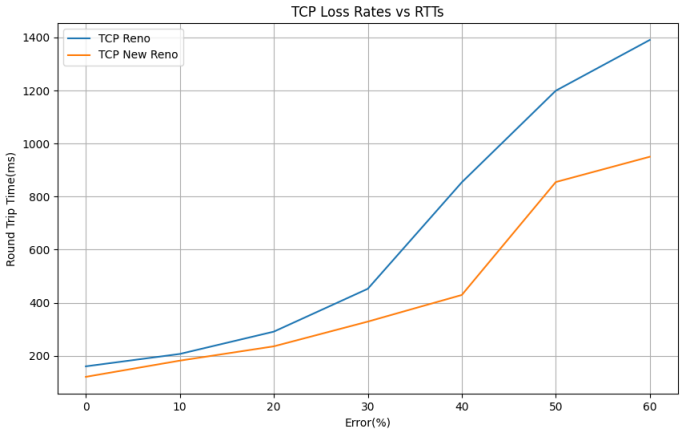

Figure 8: Comparison of RTT vs Error (%)

TCP Reno: In TCP Reno, as the error rate escalates, resulting in increased packet loss, the round-trip time (RTT) experiences a notable surge. This is primarily due to TCP Reno's reliance on timeout-based congestion control mechanisms. Upon detecting packet loss, TCP Reno promptly reduces its congestion window size, leading to congestion recovery but also causing a spike in RTT as the network readjusts.

TCP New Reno: In contrast, TCP New Reno demonstrates a more refined response to escalating error rates. Despite encountering similar increases in packet loss, TCP New Reno exhibits improved responsiveness, maintaining a relatively lower RTT. This improvement stems from TCP New Reno's capability to identify partial acknowledgments during fast recovery. By adapting its congestion window more efficiently without resorting to full timeouts, TCP New Reno effectively mitigates congestion while preserving a more stable RTT profile even under challenging network conditions.

3. **Throughput**

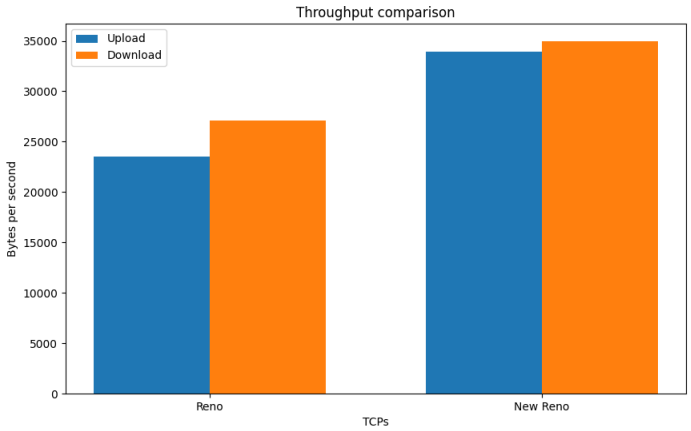

Figure 9: Comparison of Throughput

TCP Reno: Because of the reduction of CWND to half very frequently, the byte/second speed of upload and download in TCP reno is lower.

TCP New Reno: New Reno's smoother recovery mechanism, reflected in the CWND graph, is translated to a faster throughput. By avoiding unnecessary window fluctuations, New Reno potentially achieves higher average throughput compared to Reno, especially under congested network conditions.

By analyzing these comparison graphs, we can gain valuable insights into the behavior of TCP Reno and New Reno. New Reno's "recover" point mechanism can lead to smoother congestion control and potentially higher throughput compared to Reno, particularly in scenarios with higher latency or frequent packet loss.

References

[1] Geeks For Geeks. Tcp new reno.

[2] Geeks For Geeks. Tcp tahoe and tcp reno.

[3] Nidhi Kumari. Tcp reno and congestion management.

[4] Jim Kurose and Keith W. Ross. *Computer Networking: A Top-Down Approach*. Pearson; 8th edition (2020), 2020.