UNIVERSITY OF DHAKA

CSE:3111 COMPUTER NETWORKING LAB

Lab Report 6

Implementation of TCP Reno and New Reno congestion control algorithms and their performance analysis.

Shariful Islam Rayhan
Roll: 41
Md Sakib Ur Rahman
Roll: 37

Submitted to:
Dr. Md. Abdur Razzaque
Dr. Muhammad Ibrahim
Dr. Md. Redwan Ahmed Rizvee
Dr. Md. Mamun Or Rashid

Submitted on: 2024-03-16

1 Introduction

TCP Reno, one of the earliest TCP congestion control algorithms, operates by employing a combination of slow start and congestion avoidance mechanisms. When packet loss occurs, indicating network congestion, TCP Reno enters a recovery phase where it reduces its congestion window size and performs a fast retransmit of the lost packet. It then resumes its congestion avoidance phase, gradually increasing the congestion window until another congestion event occurs.

TCP New Reno builds upon the principles of TCP Reno but introduces an enhancement to better handle multiple packet losses within a single window of data. In such cases, TCP New Reno utilizes a selective acknowledgment (SACK) mechanism to accurately identify the missing segments and promptly retransmit them, rather than waiting for a timeout to occur. This improves the efficiency of recovery from multiple packet losses and helps maintain network throughput.

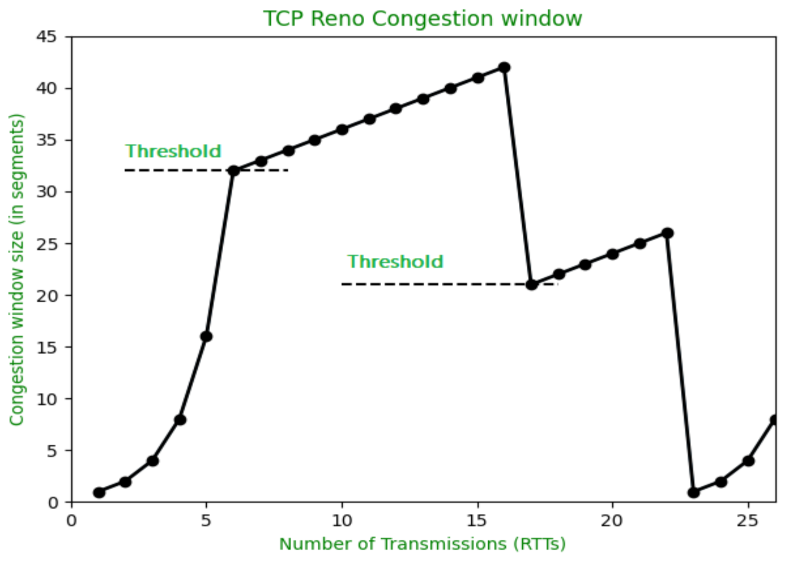

Figure 1: Graphical Analysis of TCP Reno Congestion Control Algorithm

1.1 Objectives

- Implementing tcp reno and new reno.
- See the performance analysis of both algorithm
- Comparison of both algorithm by collecting different kinds of data.

2 Theory

TCP Reno is a congestion control algorithm used in the Transmission Control Protocol (TCP) to manage network congestion. It was named after the city of Reno, Nevada, where the algorithm was first presented at a conference in 1990. TCP Reno is an extension of the earlier TCP Tahoe algorithm, and it introduces a new mechanism called and quot;fast recovery and quot; to improve network performance.

TCP Reno operates in four phases: slow start, congestion avoidance, fast retransmit, and fast recovery.

1. **Slow Start:** When a connection is established, the congestion window size is initially set to 1. The window size is increased by 1 for each ACK received, doubling the window size every round trip time (RTT) until the slow start threshold is reached.
2. **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.
3. **Fast Retransmit:** When three duplicate ACKs are received, TCP Reno assumes that a packet has been lost and immediately retransmits the lost packet
4. **Fast Recovery:**After retransmitting 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 retransmitted. After that, the congestion avoidance phase is entered, and the congestion window size is increased linearly.

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 percent 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 percent. So, for 2 packet loss, it will reduce the cwnd by 4 times (50 percent twice). But, one reduction of 50 percent 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 percent 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 startb leave alone the AIMD algorithm.

Limitations 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.

How did New Reno Evolve?

The idea was to overcome Reno's limitations. The trick was to let know the sender that it needs to reduce the cwnd by 50 percent for all the packets loss that happened in the same congestion window. This was done using partial ACK. The new type of ACK was introduced for letting know the sender about this. In Reno, after half window of silence when it receives the ACK of the retransmitted packet it considers it a new ACK and comes out fast recovery and enters AIMD then again if packet loss (same cwnd) occurs then it repeats the same procedure of reducing cwnd by half. But NewReno sender will check if the ACK 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. If all packets of that particular cwnd are ACKed by the receiver then the sender will consider that ACK as new otherwise partial ACK. If it is partial ACK then the sender will not reduce cwnd by 50 percent again. It will keep is same and won't come out of the fast recovery phase.

NewReno Solution:

It uses the concept of partial acknowledgement. When the sender receives the ACK of the first retransmitted packet then it does not consider it a "New ACK" unlike TCP Reno. NewReno checks if all the previously transmitted packets of that particular window are ACKed or not. If there are multiple packets lost in the same congestion window then the receiver would have been sending the duplicate ACKs only even after receiving the retransmitted packet. This will make it clear to the sender that all the packets are not reached the receiver and hence sender will not consider that ACK as new. It will consider it a partial ACK because only a partial window is being ACKed not the whole. Reno used to come out of the fast recovery phase after receiving a new ACK, but NewReno considers that ACK as partial and does not come out of the fast recovery phase. It wisely makes the decision of ending the fast recovery phase when it receives the Cumulative ACK of the entire congestion window.

3 Methodology

TBA

4 Experimental result

TBA

5 Experience

1. We have seen congestion control by tcp reno and tcp new reno .
2. Had an opportuinty comparison of these two algorithm .
3. It was nice to implement server and client for tcp congestion control.

References

[1] Flow Control VS Congestion Contro : `https://www.javatpoint.com/flow-control-vs-congestion-control`

[2] Flow Control and Window Size : `https://www.youtube.com/watch?v=4l2_BCr-bhw&t=177s`