University of Dhaka

Department of Computer Science and Engineering

CSE 3111 – Computer Networking Laboratory Credits: 1.5 Batch: 27/3rd Year 1st Sem 2023

Instructors: Prof. Dr. Md. Abdur Razzaque (AR), Prof. Dr. Md. Mamun-Or-Rashid (MOR),
Dr. Muhammad Ibrahim (MIb) and Mr. Md. Redwan Ahmed Rizvee (RAR)

Lab Assignment # 6

Name of the Experiment: Implementation of TCP Reno and New Reno congestion control algorithms and their performance analysis.

Objective: To understand and implement the TCP Reno and New Reno congestion control algorithms, and compare their performances.

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

1. 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.

2. 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.

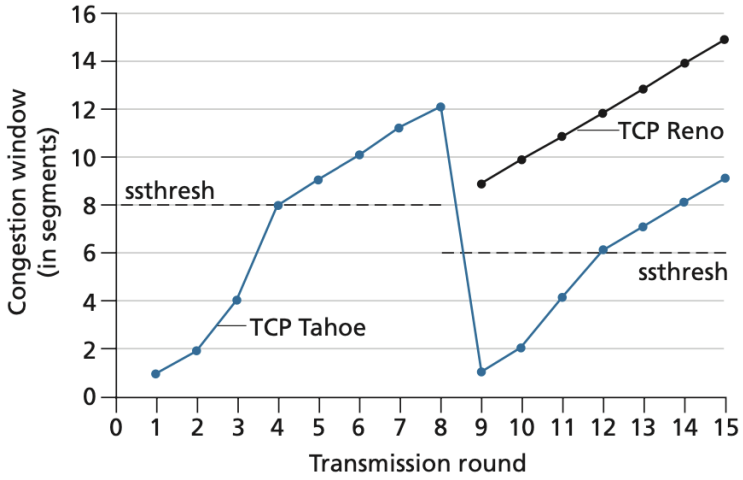

Figure 3.52 ♦ Evolution of TCP's congestion window (Tahoe and Reno)

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 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.

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% 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% 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.

Tasks :
The implementation of TCP Reno and New Reno should include the following:
- **Slow start:** Increase the congestion window size exponentially until the threshold is reached.
- **Congestion avoidance:** Increase the congestion window size linearly after the threshold is reached.
- **Fast retransmit:** Retransmit lost packets immediately after receiving three duplicate ACKs.
- **Fast recovery:** Keep the congestion window size constant after retransmitting lost packets.

Afterwards, compare the results obtained in TCP Reno and New Reno Algorithms. Analyze the differences and similarities between the two algorithms.

Deliverables:
- Source code for the client-server application with TCP Reno and New Reno
- Run simulations with TCP Reno and New Reno and collect performance metrics such as throughput, packet loss rate, and round-trip time (RTT).
- A report that documents the design and implementation of the algorithms, as well as the performance comparison under different network conditions between TCP Tahoe and TCP Reno [Do not include code in the report]

References:

- https://www.geeksforgeeks.org/tcp-tahoe-and-tcp-reno
- https://www.geeksforgeeks.org/tcp-reno-with-example/
- https://www.cs.cmu.edu/~prs/15-441-F14/lectures/rec05-congestion.pdf
- https://en.wikipedia.org/wiki/TCP_congestion_control#TCP_Tahoe_and_Reno