COL334 Computer Networks: Assignment 3

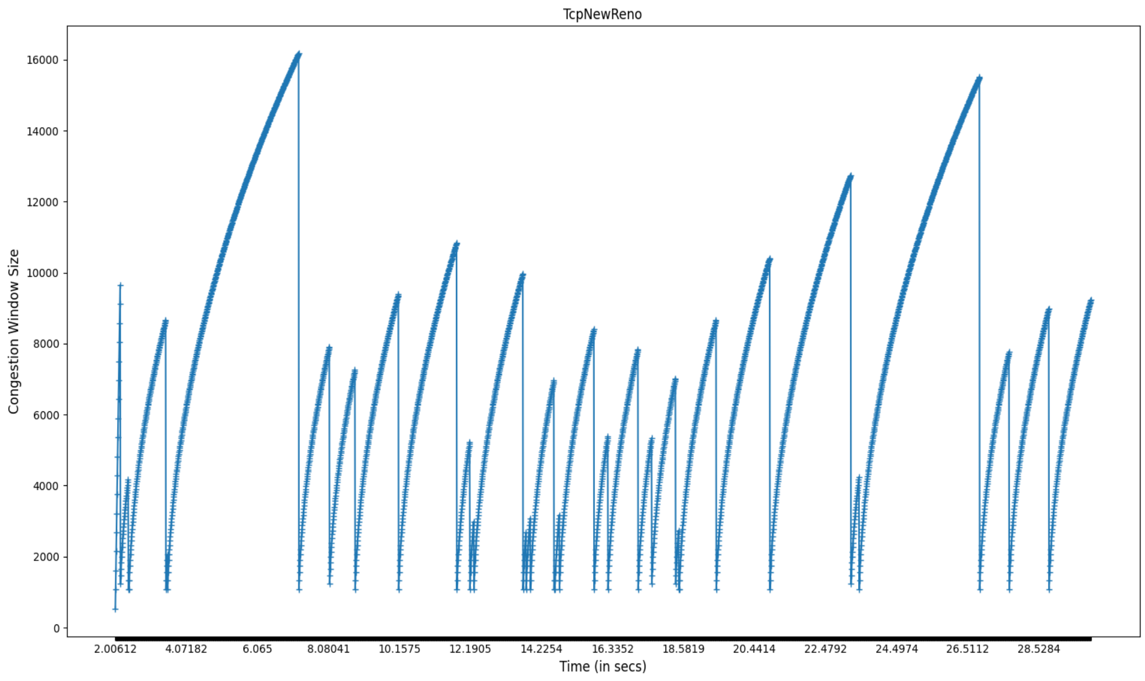
Ishita Chaudhary | 2019CS10360

Part 1: Analyzing Congestion Control Protocols

In this part, we create a simple topology of two nodes N1 and N2 and join them using a point-to-point link. We then create a TCP source at N1 and a TCP sink at N2 and fix the application data rate to be 1 Mbps. The data rate of the link between N1 and N2 is 8 Mbps, and propagation delay is 3ms. At N2, we use RateErrorModel as the error model with an error rate of 0.00001. The packet size is 3000 bytes.

Connection starts at t=1s and ends at t=30s.

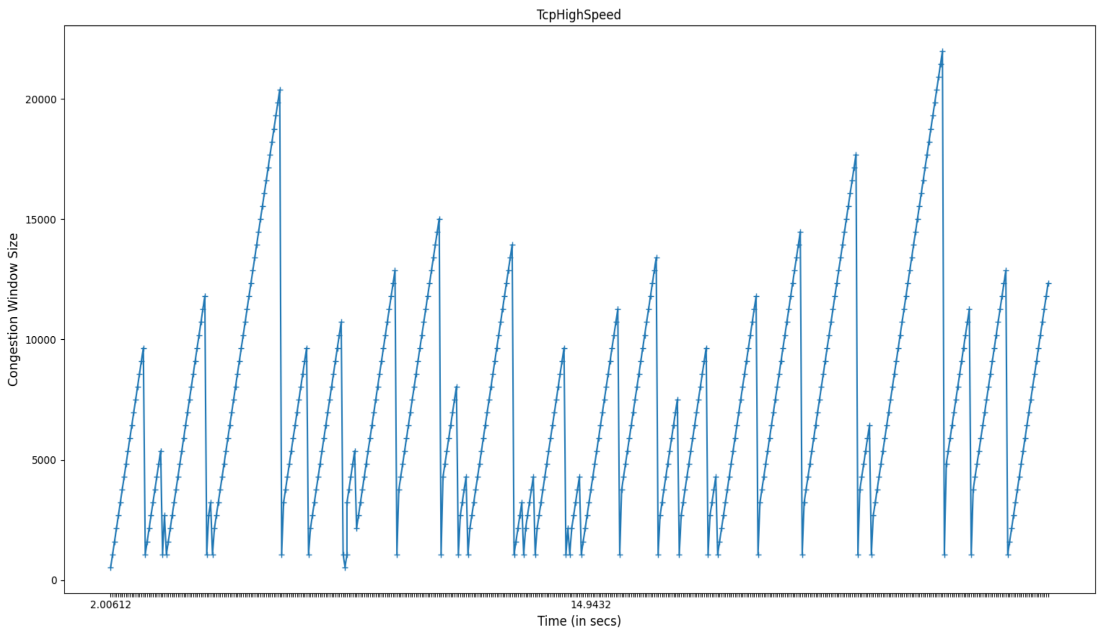
1. TcpNewReno



Total number of packets dropped = 35

Within the well-known Reno algorithm, the NewReno method introduces partial ACKs. Slow start and congestion avoidance are two possible congestion window increment approaches. Slow start begins when cwnd is updated by at most SMSS bytes for each ACK received, and slow start terminates when cwnd reaches ssthresh, i.e., congestion is detected. Because of congestion avoidance, cwnd is increased by one segment each RTT, and the slow start threshold is reduced to half for each congestion occurrence. The graph rises to a certain point before descending sharply.

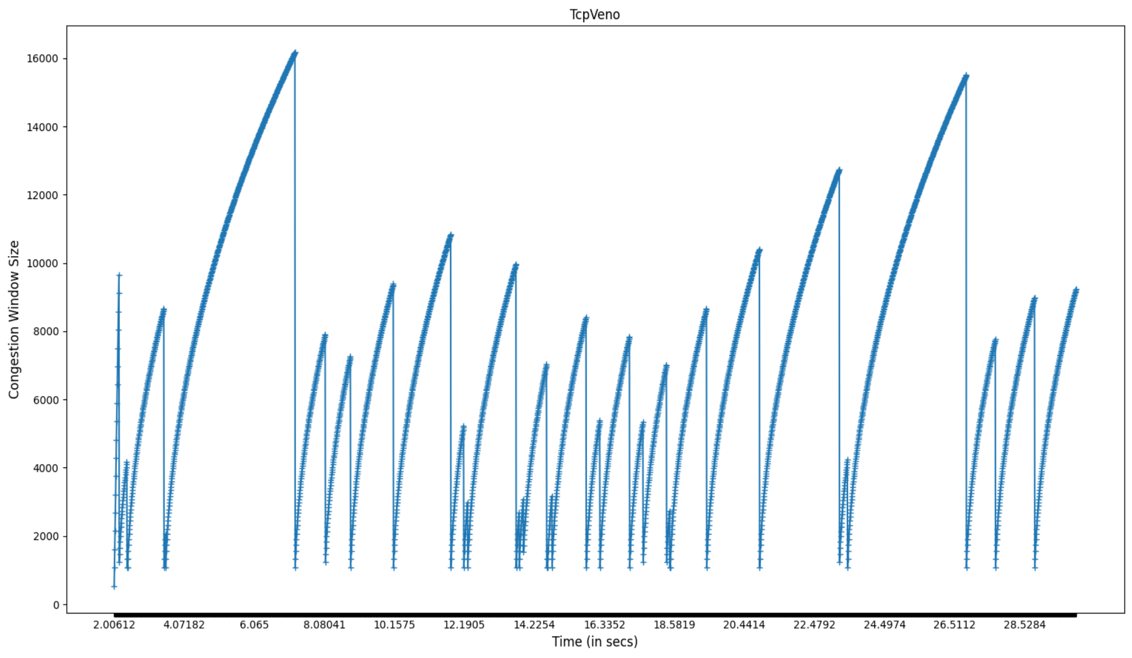
2. TcpHighSpeed



Total number of packets dropped = 35

For TCP connections with high congestion windows, the HighSpeed congestion protocol was devised. When the window crosses a particular threshold, HighSpeed causes the cwnd to increase quicker than the prior protocol, as shown in the graph above. We know that cwnd += a(cwnd)/cwnd, where a is a function given in RFC. The slow start threshold is reduced when a congestion event occurs, and the congestion window is set to cwnd = (1-b(cwnd)). cwnd, where b is an RFC-derived function. The graph shows a relatively low frequency of saturation. As the window size approaches saturation, the rate declines. During the probing phases, HighSpeed causes the cWnd to increase rapidly.

3. TcpVeno



Total number of packets dropped = 35

TCP Veno uses Vegas's approach to estimate the backlog in the bottleneck queue to distinguish between congested and non-congested states and is an improvised version of the Reno algorithm and its ability to successfully deal with random packet loss in wireless access networks. The peaks are rather numerous, and some of them are larger than the others.

The graph also certainly looks like the NewReno protocol we looked at before. The RTT and its preset threshold beta are used to determine the cwnd modification choice.

Mathematically speaking,

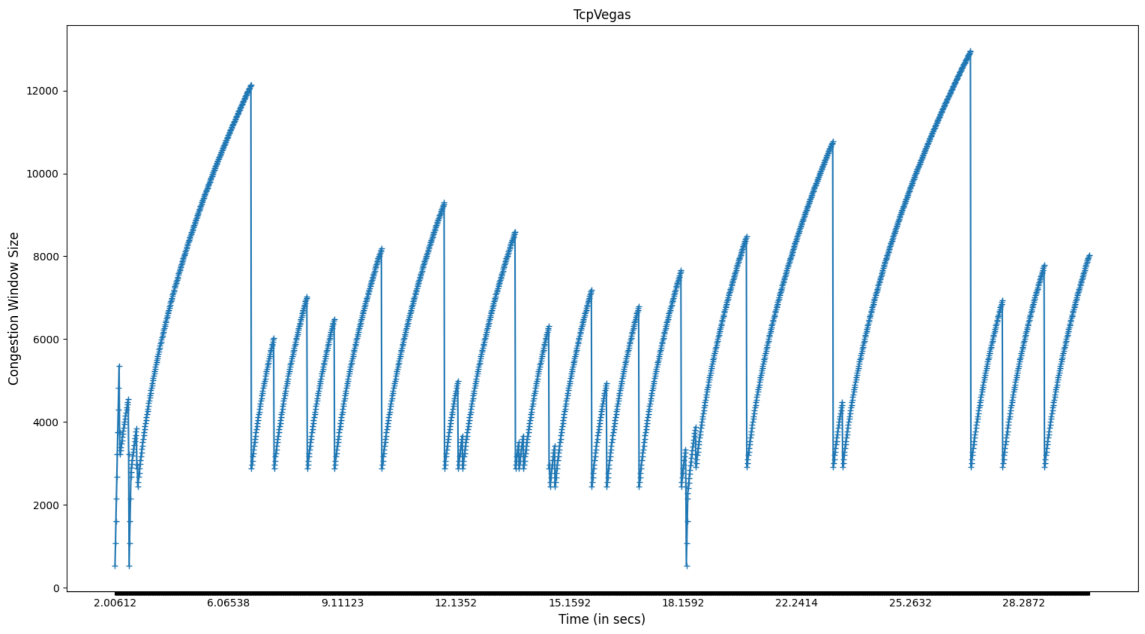
N = Actual (RTT- BaseRTT)

= Diff. BaseRTT

Diff = Expected - Actual = (cwnd/BaseRTT)-(cwnd/RTT)

In general, the connection remains steady for prolonged. The highest congestion window size achieved was fairly comparable to the NewReno method, which is less than HighSpeed but much larger than the Vegas algorithm.

4. TcpVegas



Total number of packets dropped = 35

TCP Vegas is a delay-based congestion control algorithm that uses a strategic approach to avoid packet losses by keeping a slight backlog in the bottleneck queue. This protocol has a greater packet loss rate than the others. It decreases the window size after experiencing a packet drop, which is much larger than the rest of the protocols. However, packet loss is higher than the other protocols.

Mathematically speaking,

Diff = Expected - Actual = (cwnd/BaseRTT)-(cwnd/RTT)

After switching from the slow start mode, this protocol linearly changes its congestion window to avoid traffic.

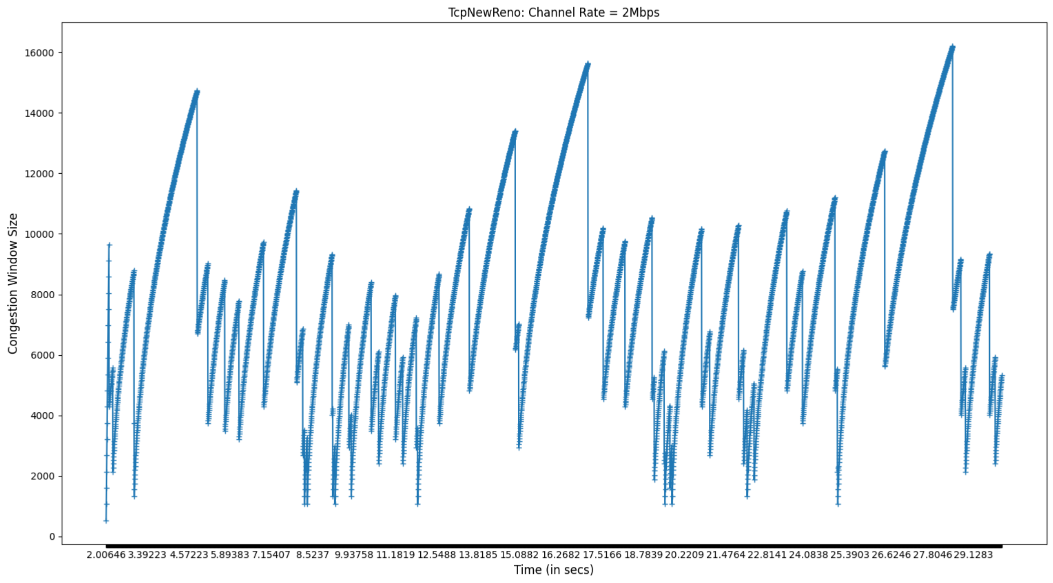
Packet Dropping: We found that the NewReno, HighSpeed, and Veno algorithms, as well as the Vegas algorithm, all had the same amount of packet losses. The Vegas algorithm plot differs from the other three in some ways. However, all four methods drop around the same amount of packets in nearly the same time interval, with all other circumstances being identical.

Part 2: Effect of the Bandwidth/ Application Data Rate on Congestion Window Size

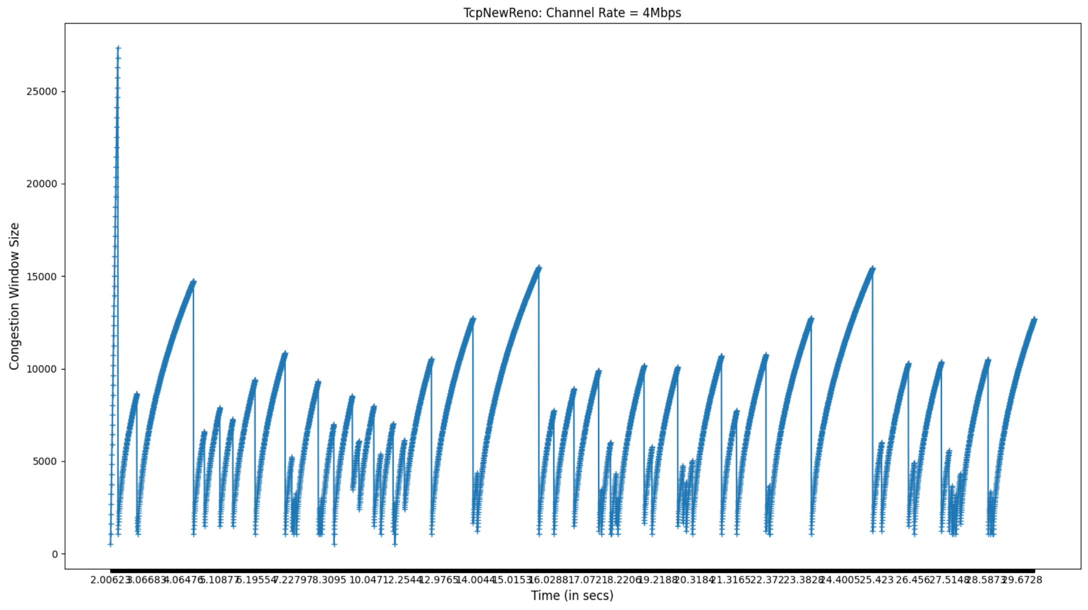
In this part, we create a simple topology of two nodes N1 and N2, and join them using a point-to-point link. Then we create a TCP source at N1 and a TCP sink at N2. We use TcpNewReno as the congestion protocol for all the parts.
The channel delay is 3ms. At N2, we use RateErrorModel as the error model with an error rate of 0.00001. The connection starts at t=1s and ends at t=30s. The packet size is 3000 bytes.

1. Varying Channel Data Rate

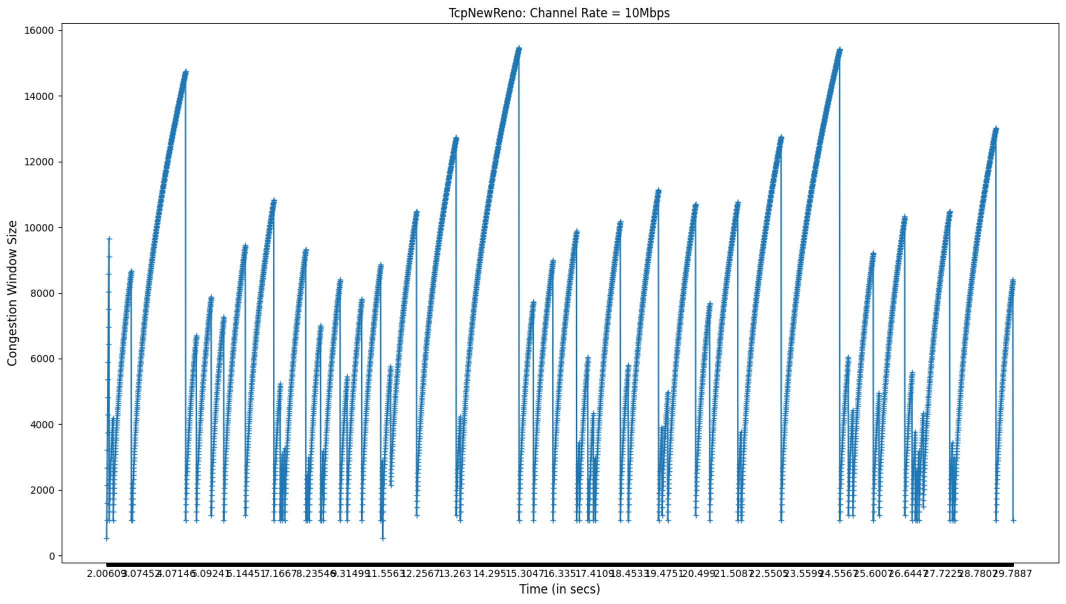
We will observe the congestion at different Channel Data Rates (2Mbps, 4Mbps, 10 Mbps, 20Mbps, 50 Mbps), keeping the Application Data Rate constant and equal to 2Mbps.



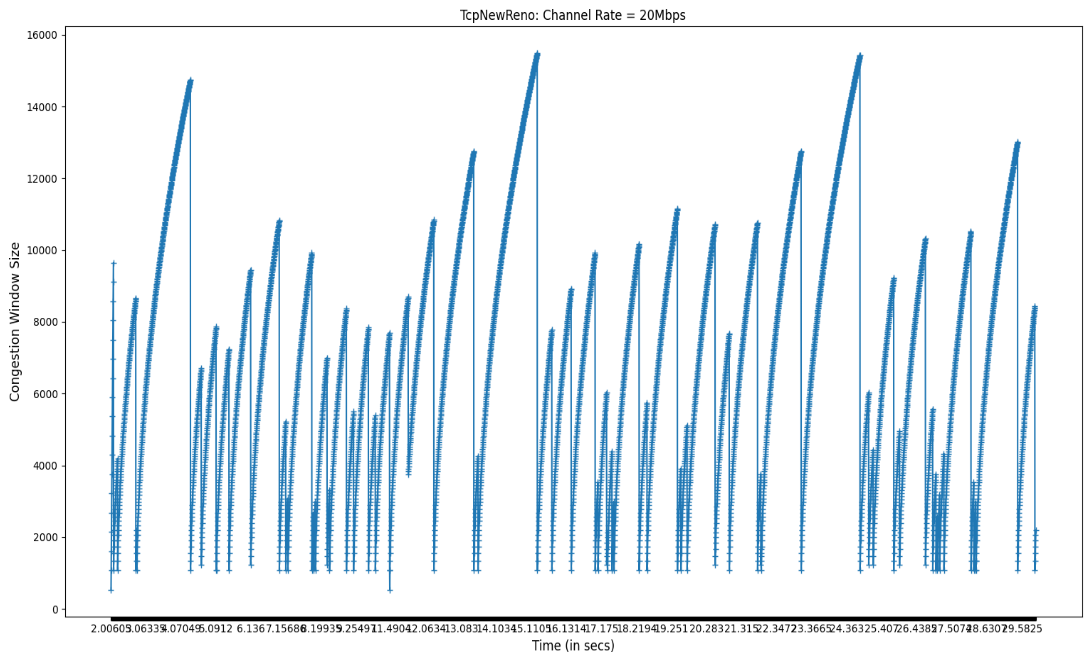
Total number of packets dropped = 60



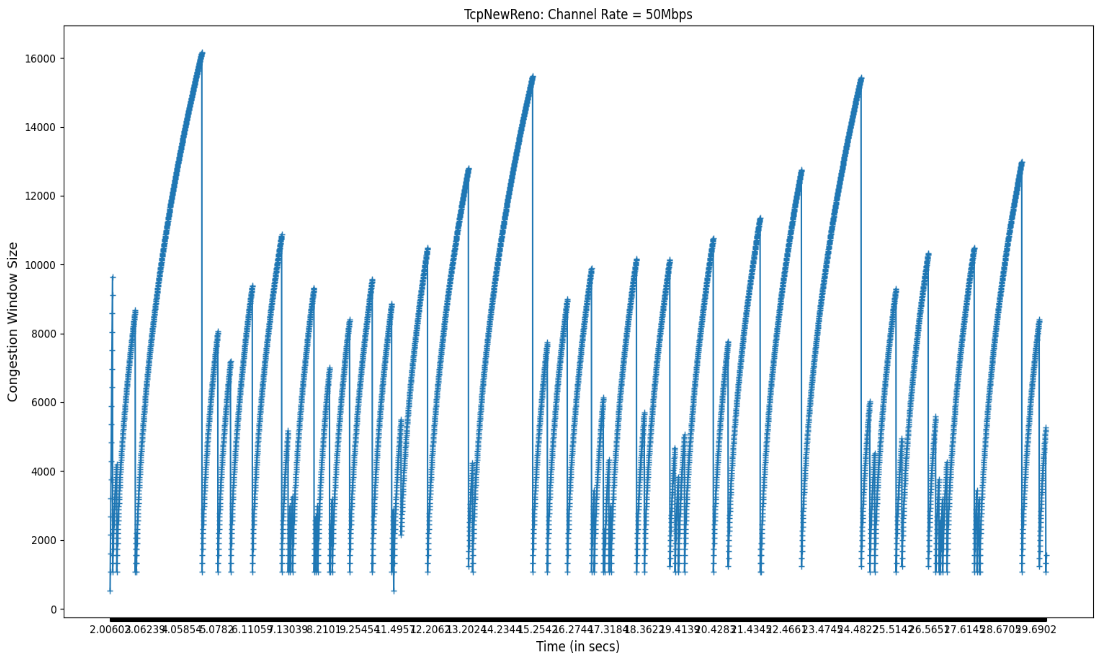
Total number of packets dropped = 68



Total number of packets dropped = 70



Total number of packets dropped = 71



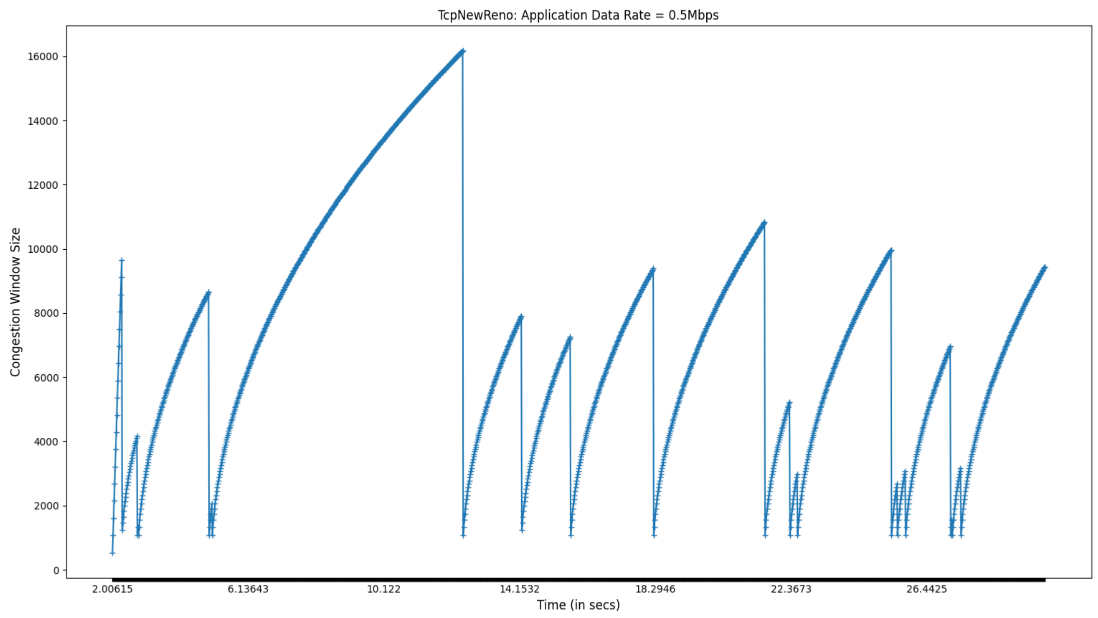
Total number of packets dropped = 73

Channel Rate	2 Mbps	4 Mbps	10 Mbps	20 Mbps	50 Mbps
Packet Loss	60	68	70	71	73

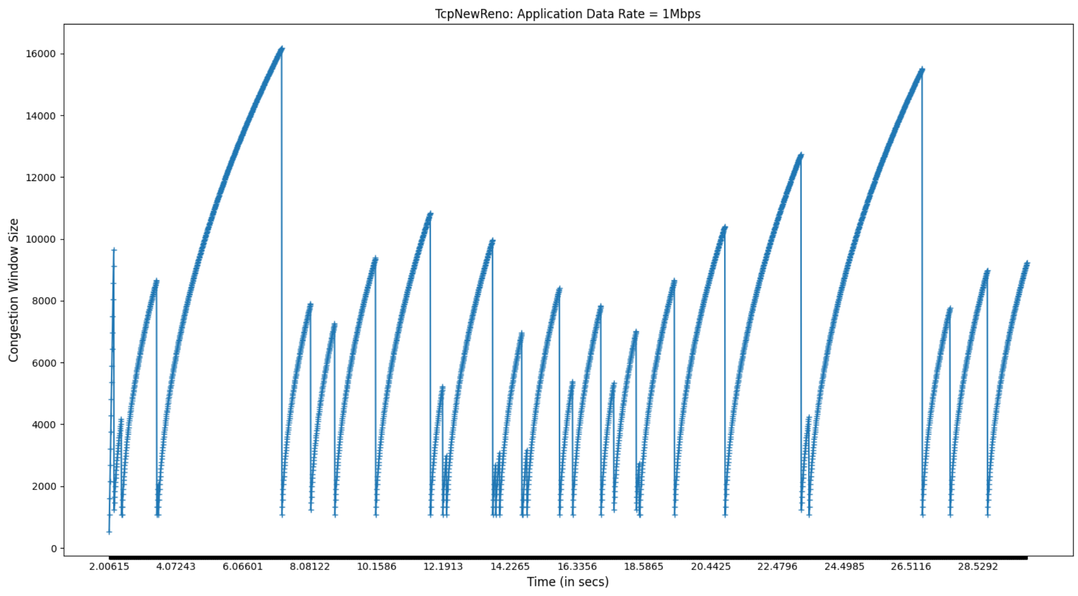
The number of packets missed grew from 60 to 73 as the Channel data rate increased from 2Mbps to 50Mbps. Over the five graphs shown above, the maximum congestion window size does not change significantly, except the first peak of 4Mbps, which is significantly greater than the others.. The graphs for the last three data rates, i.e., 10, 20, and 50 Mbps are quite similar, showing that that range has reached saturation. In addition, the number of packets lost also appears to have reached a saturation point, since it only grows by one or two over the final three graphs.
This might have happened because the application rate is 6Mbps, any channel rate over that figure has no discernible effect on the graphs. In comparison to the graphs below 6Mbps, i.e. 2Mbps and 4Mbps, have a greater frequency.

2. Varying Application Data Rate

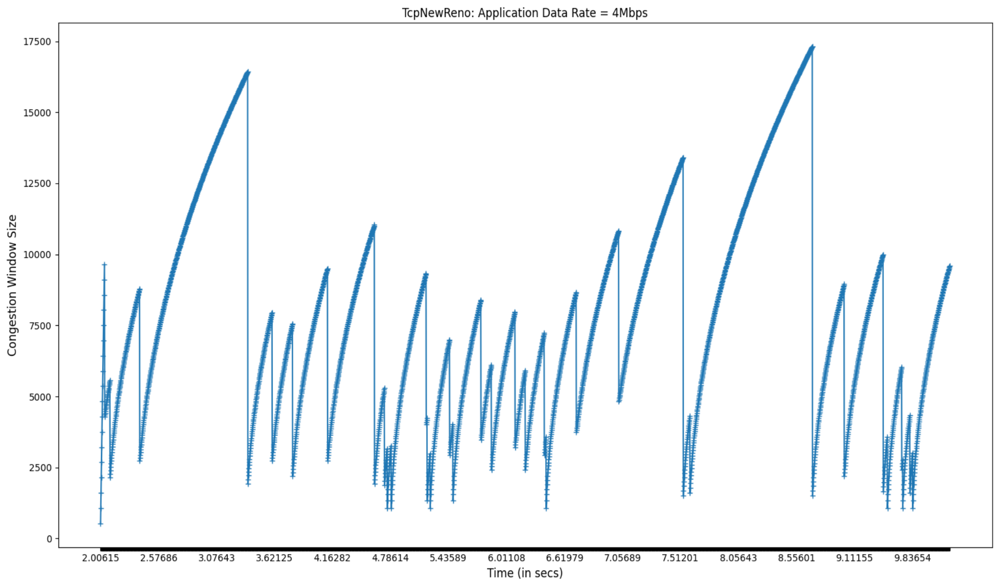
We observe the congestion at different Application Data Rates (0.5Mbps, 1Mbps, 2Mbps, 4Mbps, 10Mbps), keeping the Channel Data Rate constant and equal to 6Mbps.



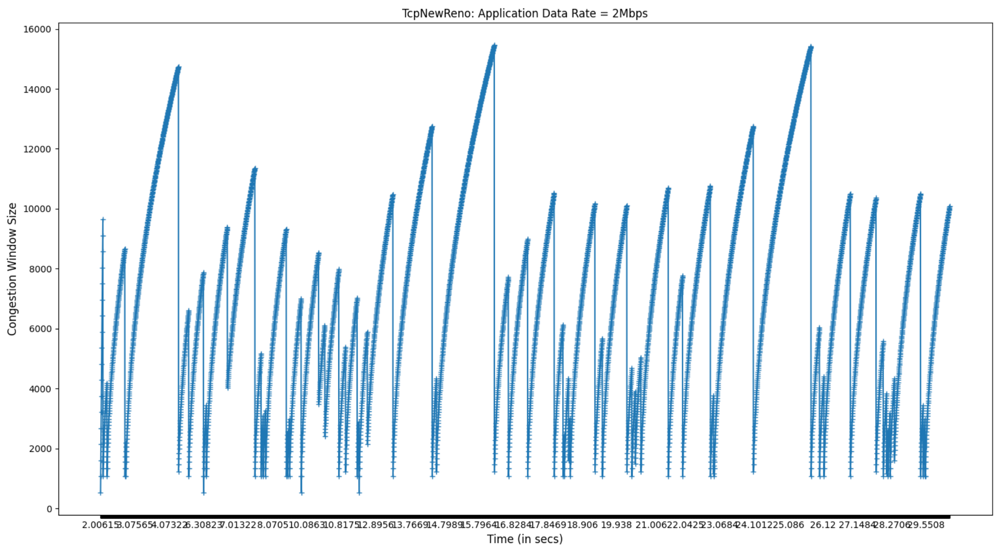
Total number of packets dropped = 21



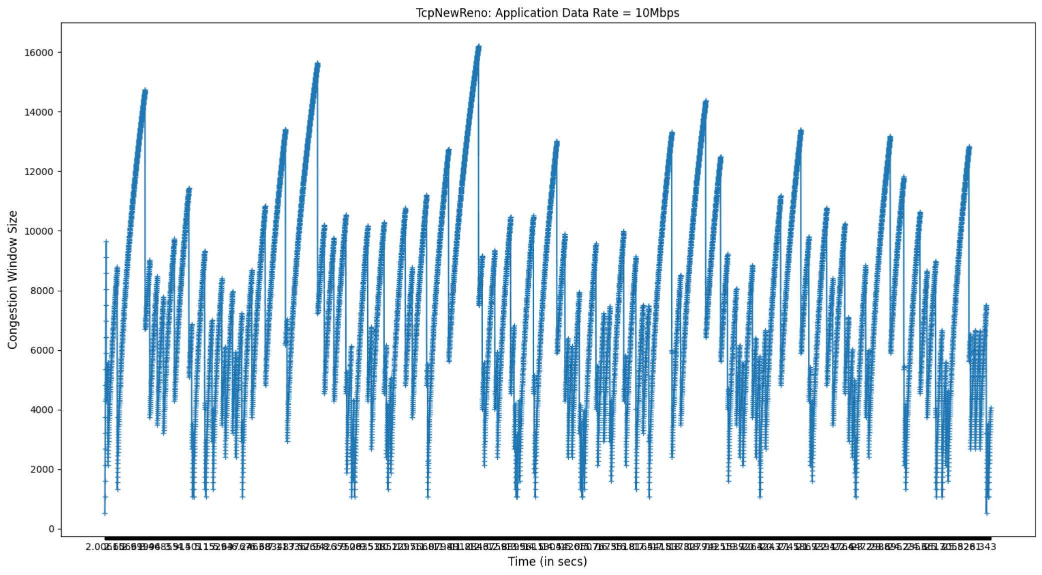
Total number of packets dropped = 35



Total number of packets dropped = 67



Total number of packets dropped = 140



Total number of packets dropped = 162

App. Data Rate	0.5 Mbps	1 Mbps	2 Mbps	4 Mbps	10 Mbps
Packet Loss	21	35	67	140	162

More packets drop during the same time interval when the Application data rate increases while the Channel data rate remains unchanged. At a rate of 0.5Mbps, the number of packets lost was 21, which grew to 162 at 10Mbps. We notice that as the Application data rate increases, the plot becomes more condensed, implying that it is stable over shorter time periods. The maximum congestion window sizes are almost the same in the five graphs above. Since they reach the threshold so promptly, graphs with greater application rates have a higher frequency.