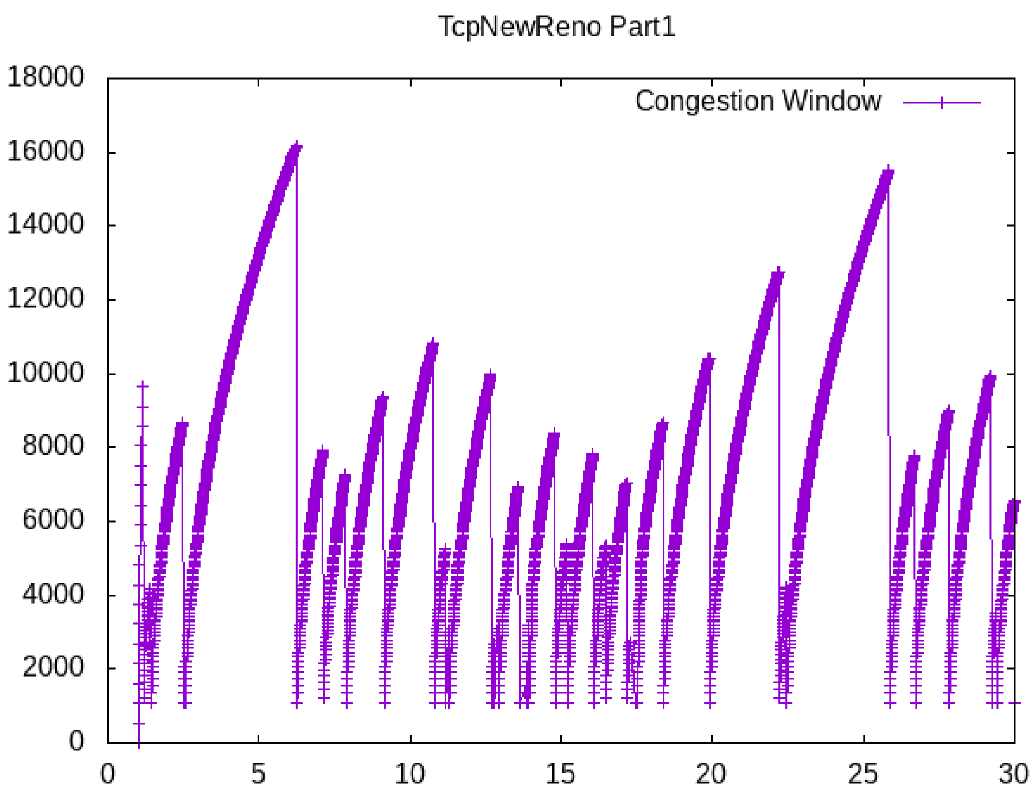
COL334 ASSIGNMENT 2

ARPIT CHAUHAN - 2019CS10332

October 26, 2021

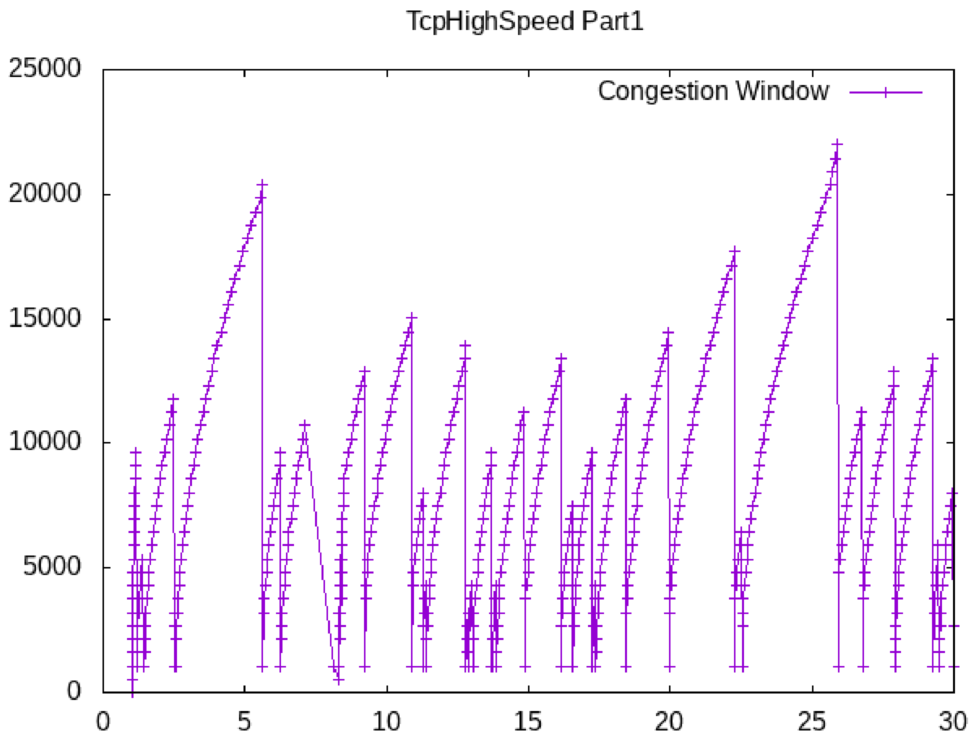
- ## PART 1

1. 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. We 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 error rate of 0.00001. Packet size is 3000 bytes.



Total number of packets dropped = 38

The NewReno algorithm introduces partial ACKs inside the well-established Reno algorithm. Two possible congestion window increment strategies are Slow start and congestion avoidance.
During slow start, cwnd is incremented by at most SMSS bytes for each ACK received and when cwnd exceeds ssthresh i.e. congestion is observed, slow start ends.
During congestion avoidance, we increment cwnd by 1 segment size per RTT and whenever a congestion event is observed, slow start threshold is halved.

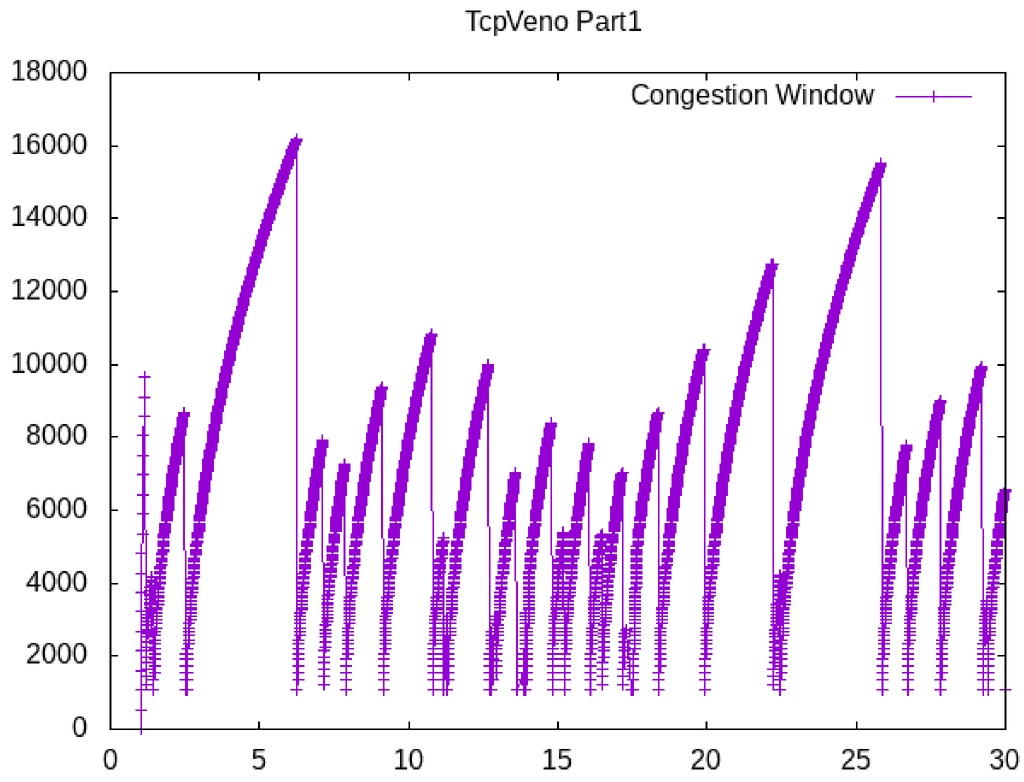


Total number of packets dropped = 38

The HighSpeed congestion protocol has been designed for TCP connections with large congestion windows.

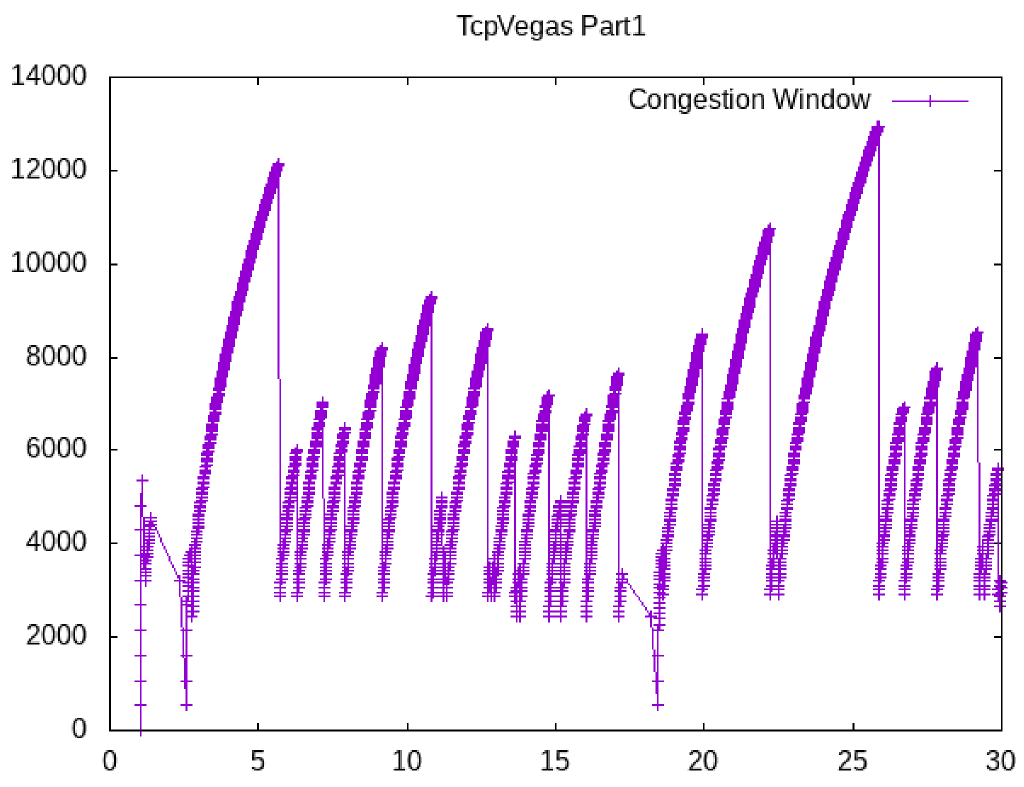
We can observe from the plot above that HighSpeed makes the cwnd grow faster than the previous protocol when the window exceeds a certain threshold.

We know that mathematically, cwnd += a(cwnd)/cwnd where a is a function mentioned in RFC. When there is a congestion event, the slow start threshold is decremented and the congestion window is set to cwnd = (1-b(cwnd)).cwnd where b is a function taken from RFC.



Total number of packets dropped = 38

This algorithm enhances the Reno Algorithm so that random packet losses in wireless access networks can be dealt more effectively.
The plot is also very similar to the NewReno protocol studied earlier.
It makes the decision on cwnd modification using the RTT and its predefined threshold beta. Generally, the connection stays longer in stable state.
The maximum congestion window size reached in very similar to the NewReno, lesser than HighSpeed and considerably larger than the Vegas algorithm.



Total number of packets dropped = 39

Different from the above two protocols, TCP Vegas is a pure delay-based congestion control protocol that tries to prevent packet drops by maintaining a small backlog at the bottleneck queue.
From the plot above, we can observe that the maximum congestion window is lesser than the 3 congestion control protocols observed before.
To avoid congestion, this protocol linearly increases/decreases its congestion window, after switching from the slow start mode.

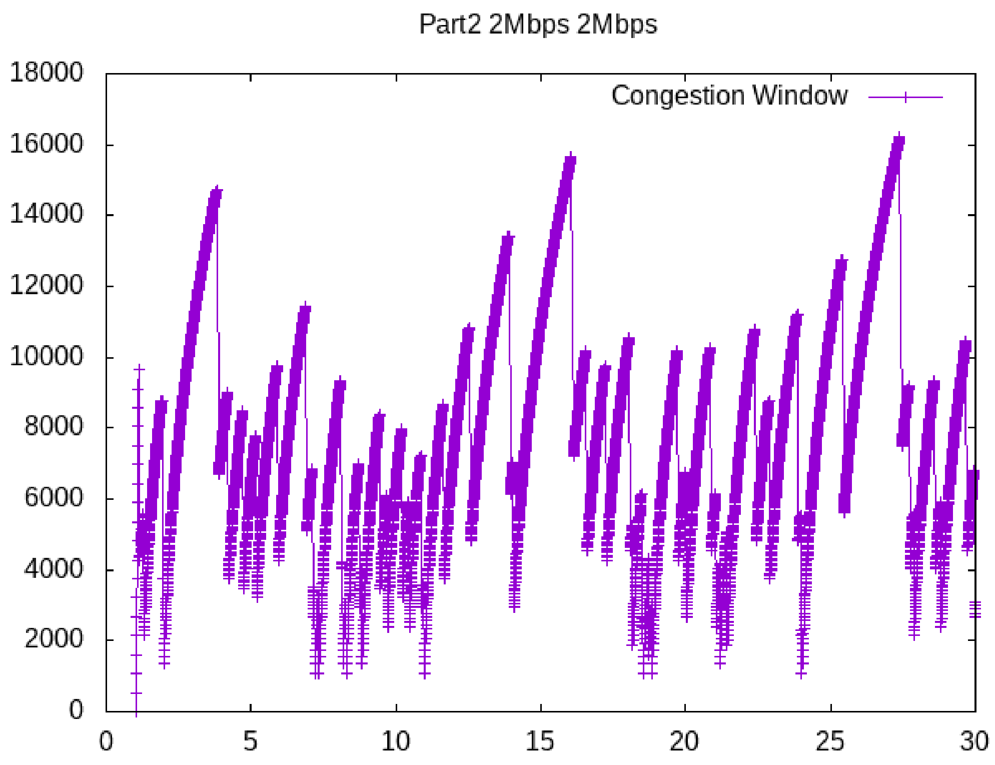
PACKET DROPPING

We observe that same number of packet drops were observed for NewReno, HighSpeed and Veno algorithm whereas 1 extra packet was dropped for the Vegas algorithm. The plot for Vegas algorithm is also somewhat different from the other three, but more or less all the four algorithms have similar number of packets being dropped in the same time interval, with all other conditions also being kept same.

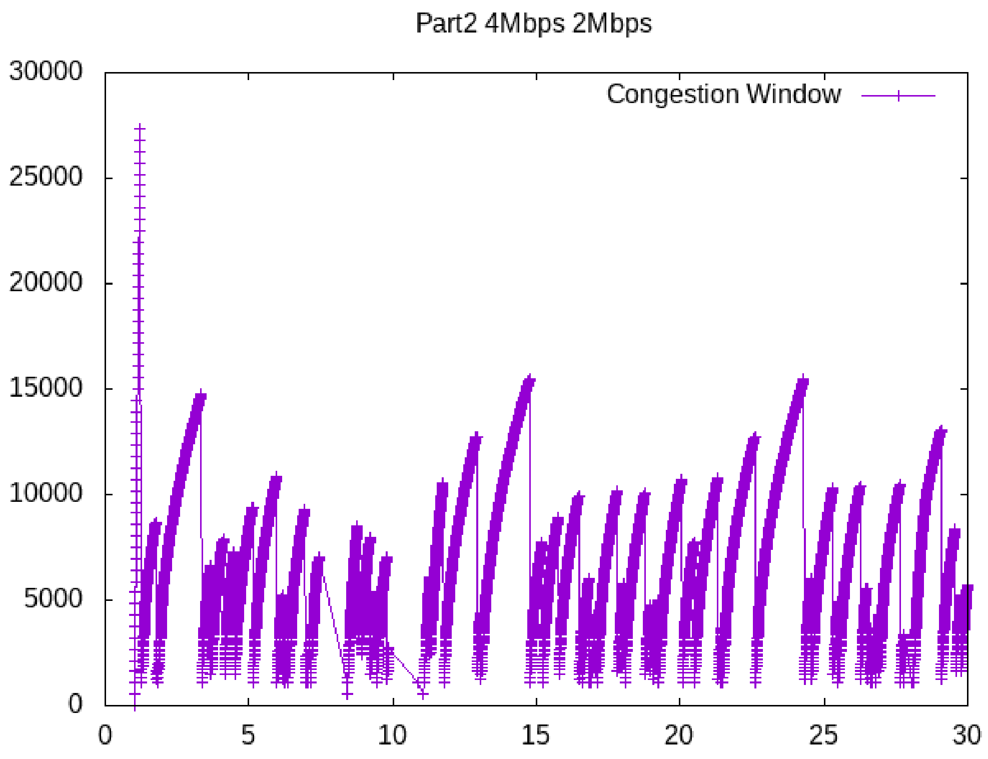
- **PART 2**

  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. The channel delay is 3ms. At N2, we use RateErrorModel as the error model with error rate as 0.00001. The connection starts at t=1s and ends at t=30s. Packet size is 3000 bytes.

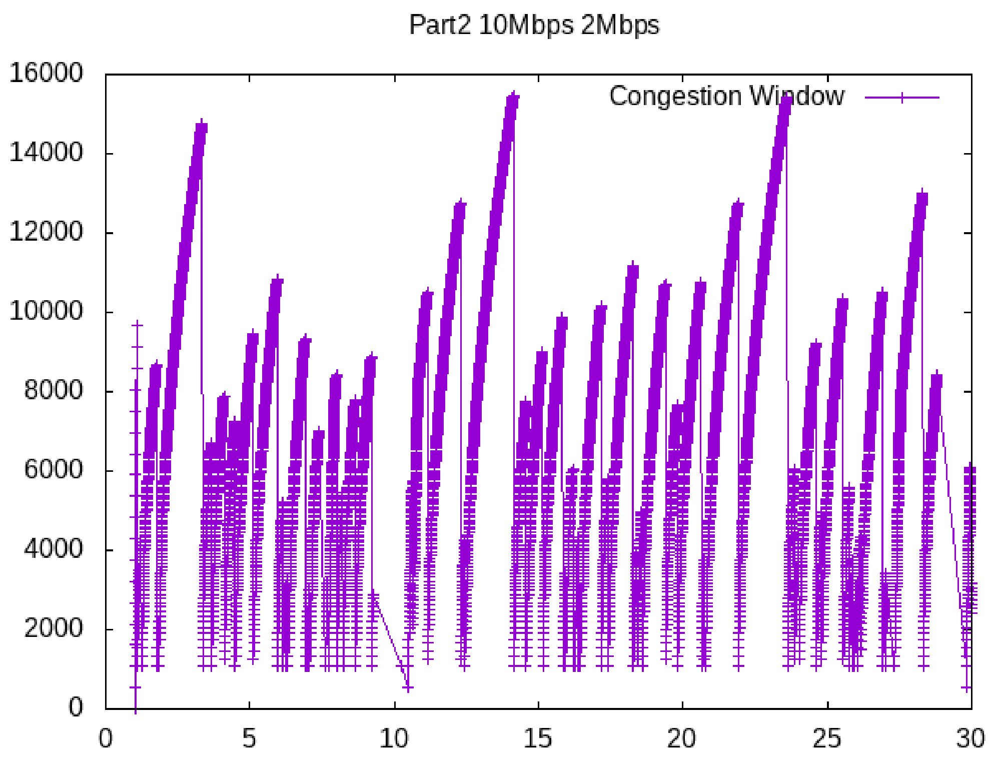
  A) Now, we observe the congestion at different Channel Data Rates (2Mbps, 4Mbps, 10 Mbps, 20Mbps, 50 Mbps) keeping the Application Data Rate as 2Mbps .



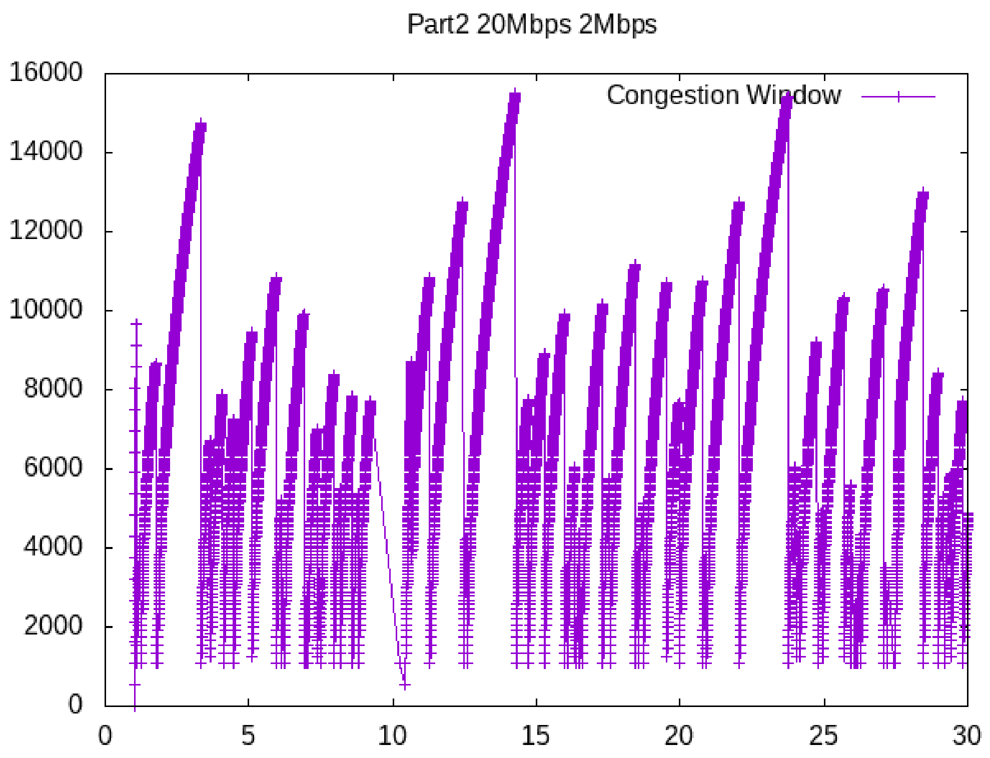
Total number of packets dropped = 62



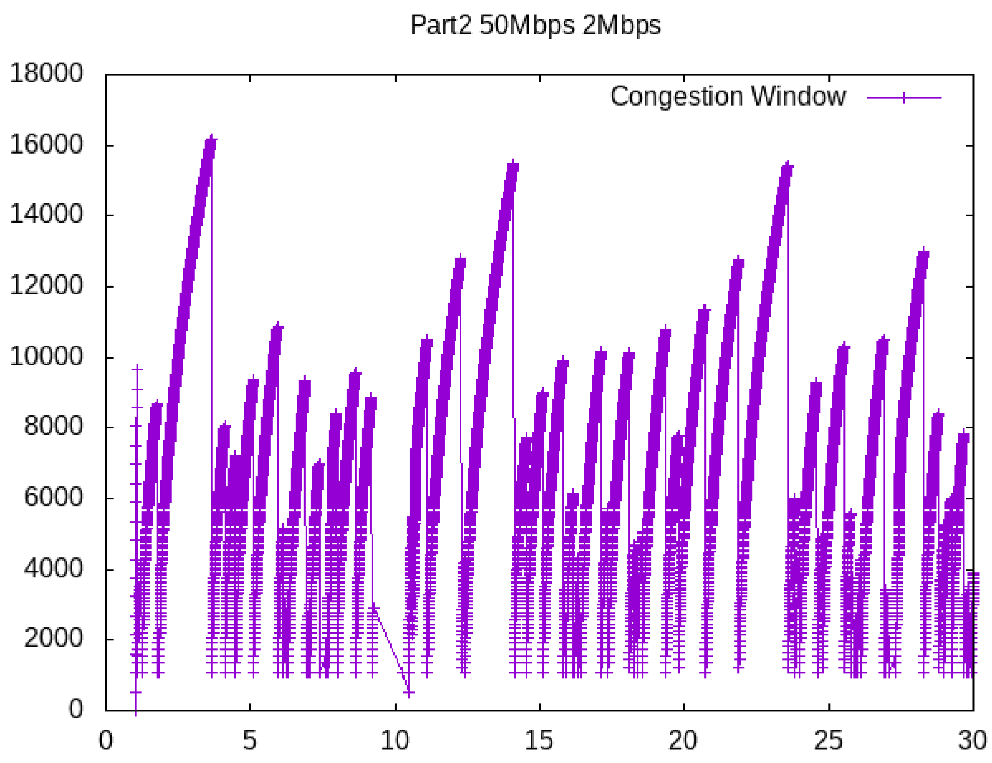
Total number of packets dropped = 72



Total number of packets dropped = 73



Total number of packets dropped = 74



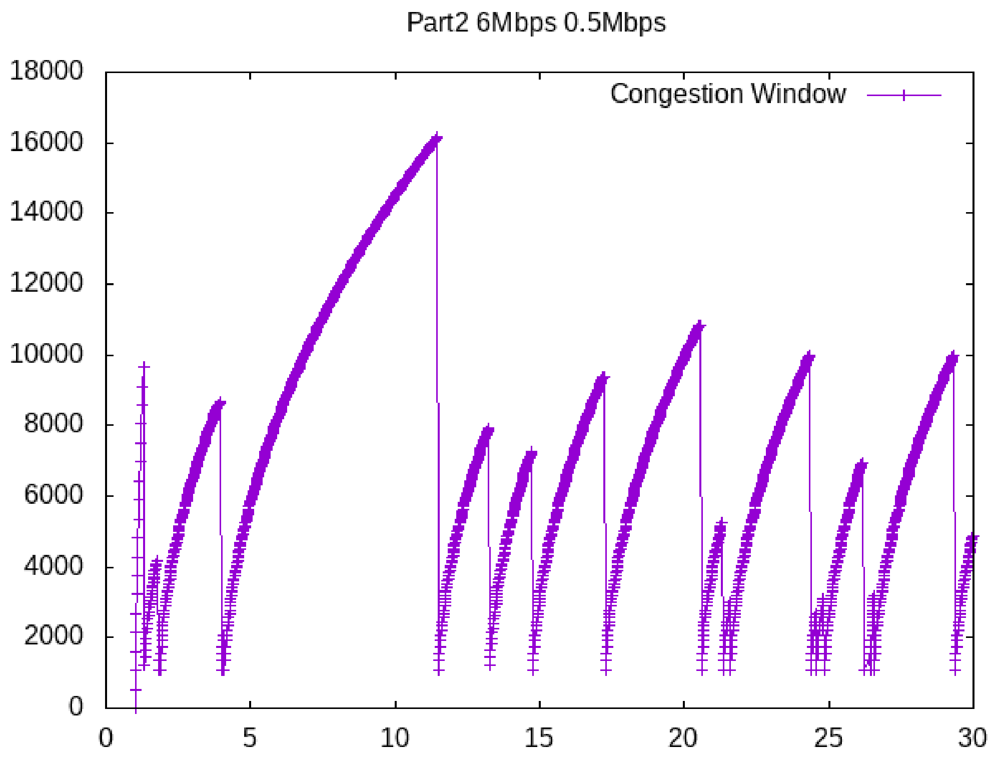
Total number of packets dropped = 75

We observe that as the Channel data rate is increased keeping the application data rate constant, more number of packets dropped during the same time interval.As we increased the Channel data rate from 2Mbps to 50Mbps the number of packets dropped increased from 62 to 75. The maximum congestion window size does not change by much over the five plots given above.

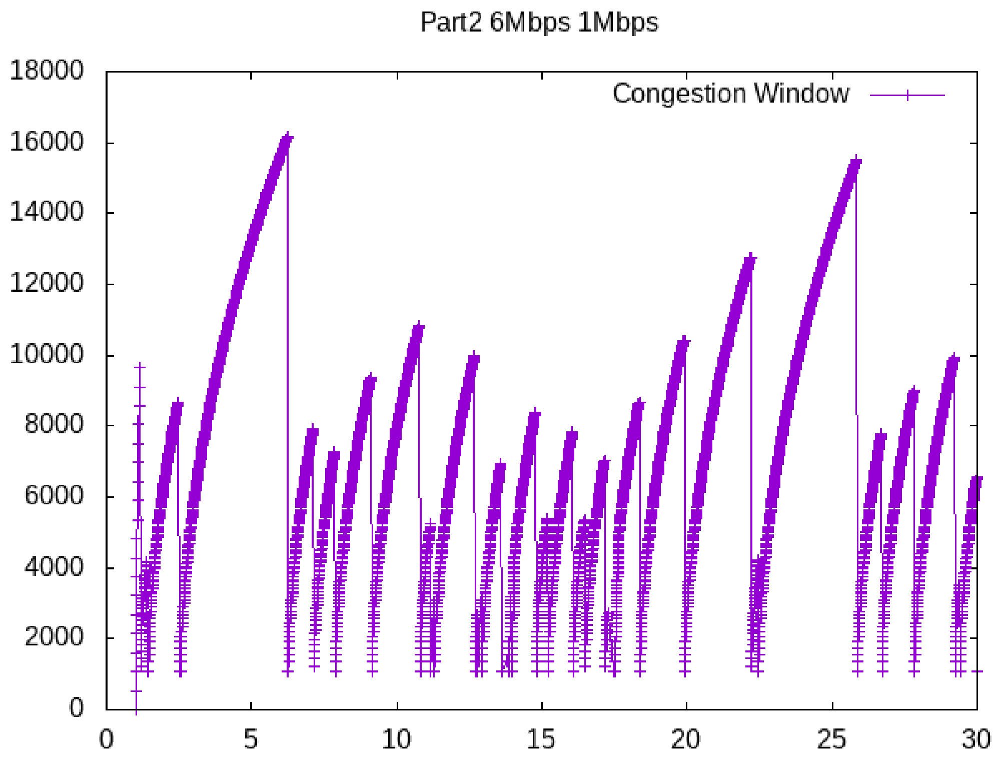
The plots for 10Mbps and 20Mbps are very similar implying some sort of saturation at that range. Also the number of packets dropped seem to saturate as it increases only by 1 for the last 3 plots.

We also observe that the initial peak observed for 4Mbps is exceptionally higher than the others.

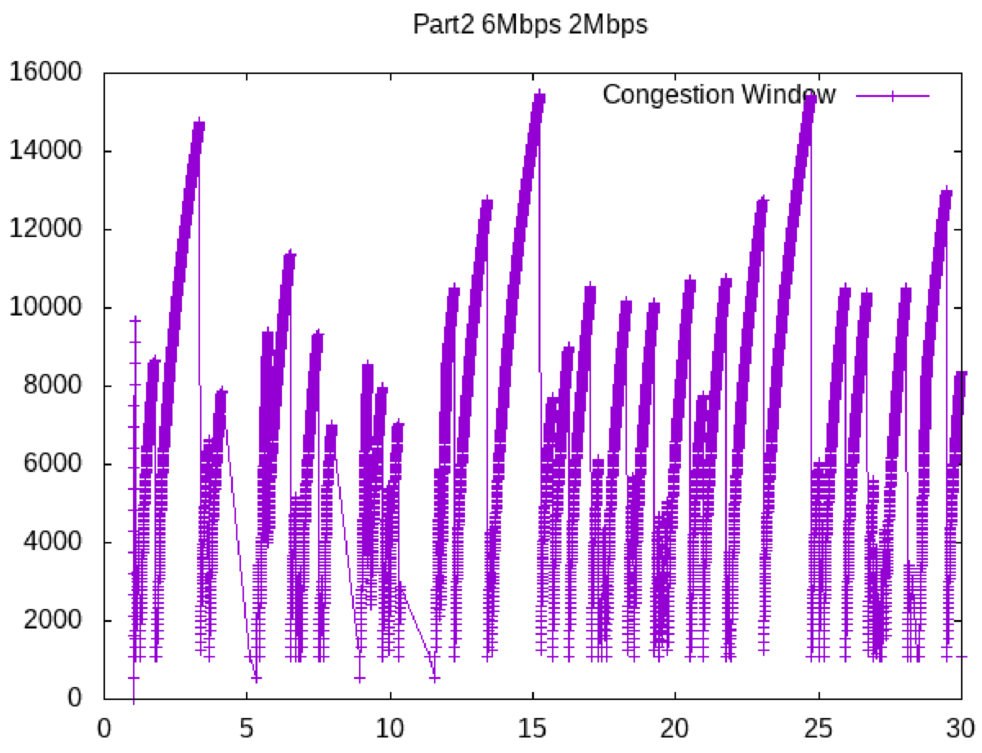
B) Now, we observe the congestion at different Application Data Rates (0.5Mbps, 1Mbps, 2Mbps, 4Mbps, 10Mbps) keeping the Channel Data Rate as 6Mbps.



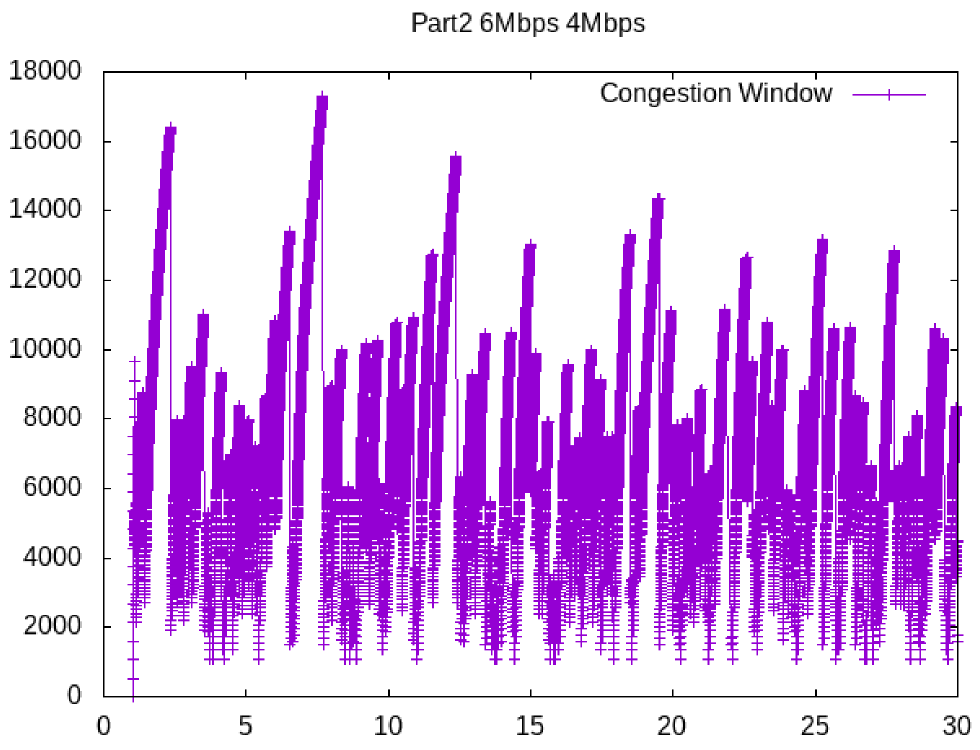
Total number of packets dropped = 22



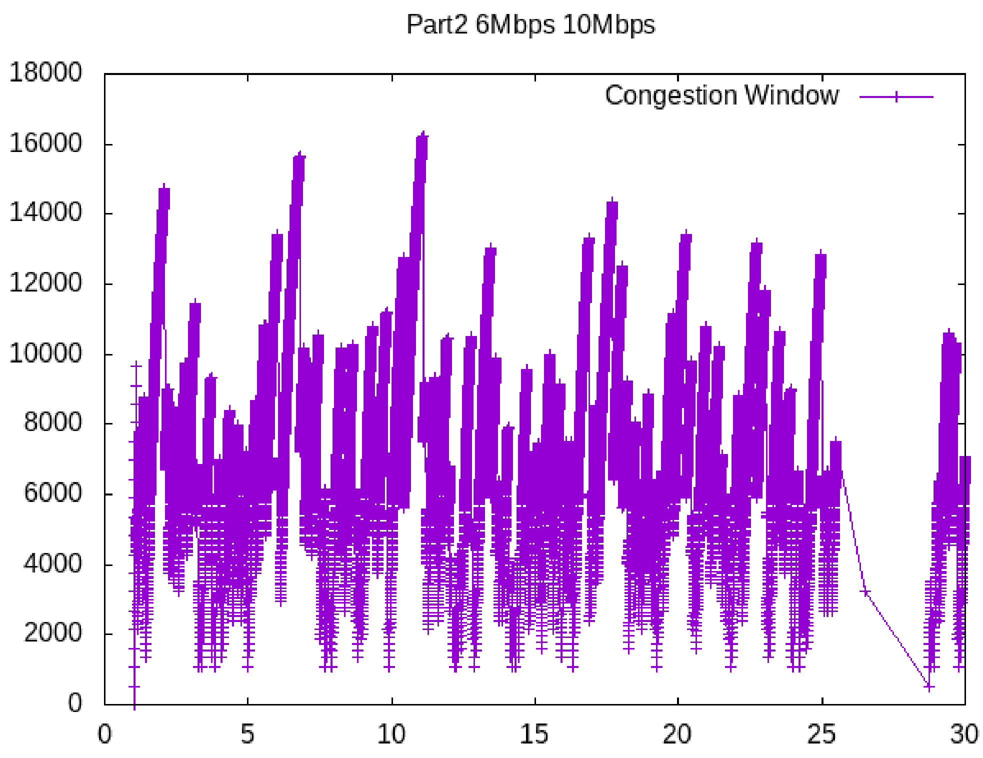
Total number of packets dropped = 38



Total number of packets dropped = 71



Total number of packets dropped = 156



Total number of packets dropped = 156

We observe that as the Application data rate is increased keeping the Channel data rate constant, more number of packets dropped during the same time interval. The number of packets dropped increased from 22 at ADR = 0.5Mbps to 156 at both 4Mbps and 10Mbps. Also the plots for 4Mbps and 10Mbps are very similar which implies some sort of saturation as the Application Data rate is increased.
As the Application data rate is increased, we observe that the plot keeps getting more and more compressed which implies that it is stable for smaller durations of time.
The maximum congestion window sizes are more or less in the same range across the five plots given above just that their timings are considerably different.

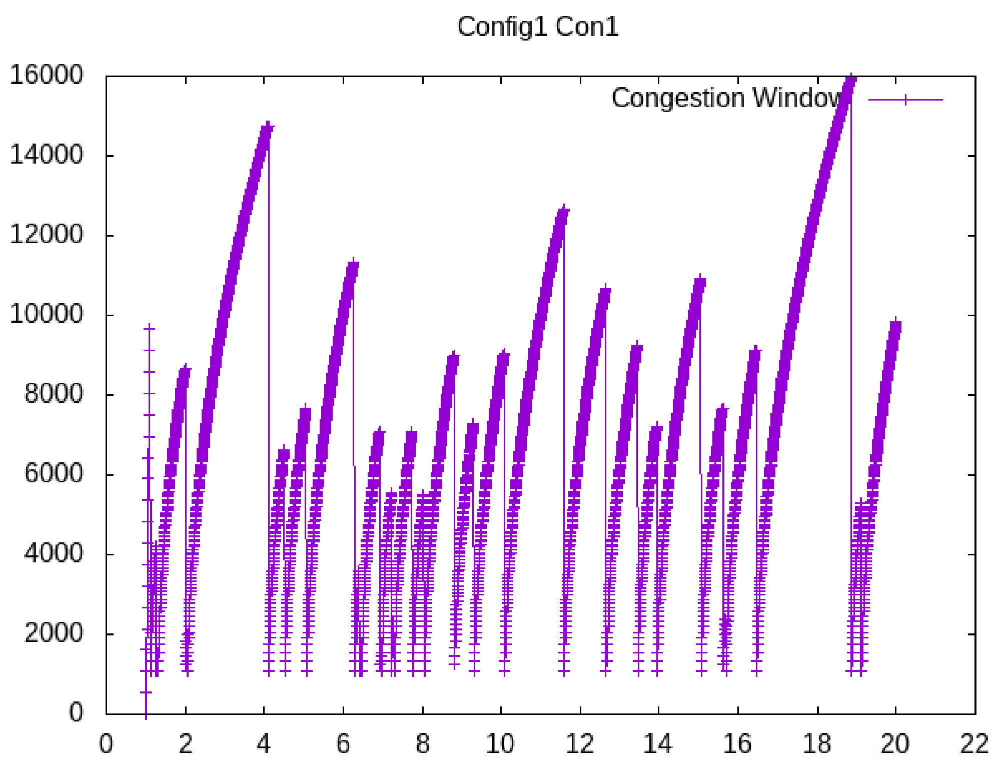
- **PART 3**

  In this part, we have 3 nodes and 3 connections with TCP sources at N1 and N2 and TCP sink at N3. N1-N3 and N2-N3 are connected via point-to-point link. Other specifications have been specified in the problem statement.

CONFIGURATION 1

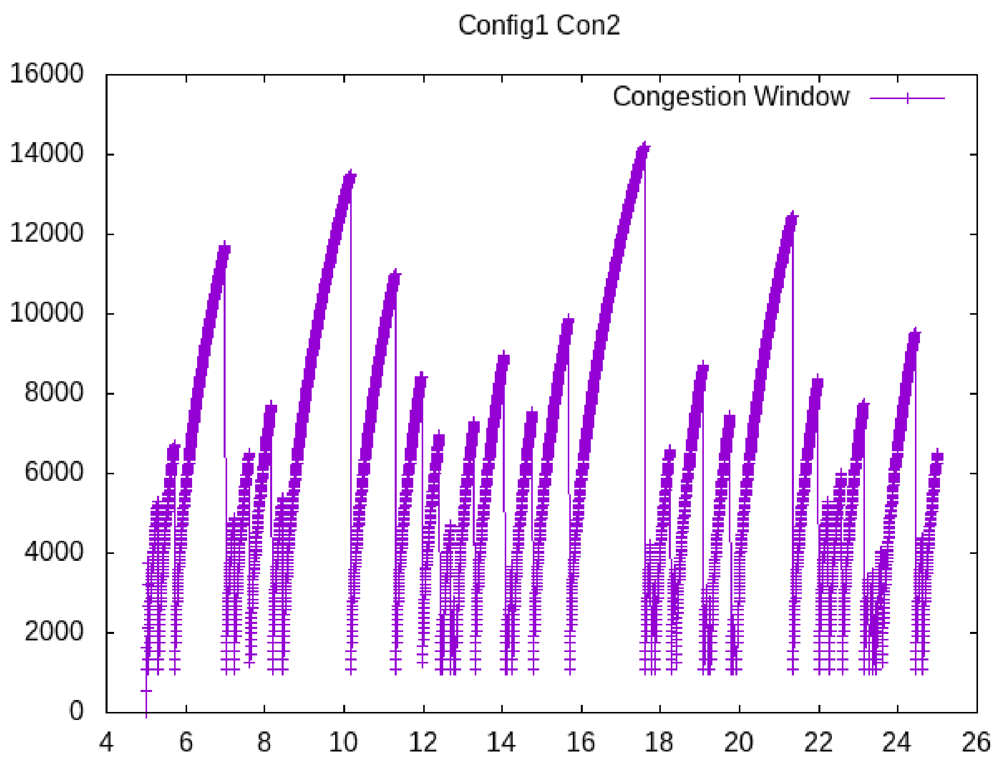
In this configuration, all the Senders use TcpNewReno.

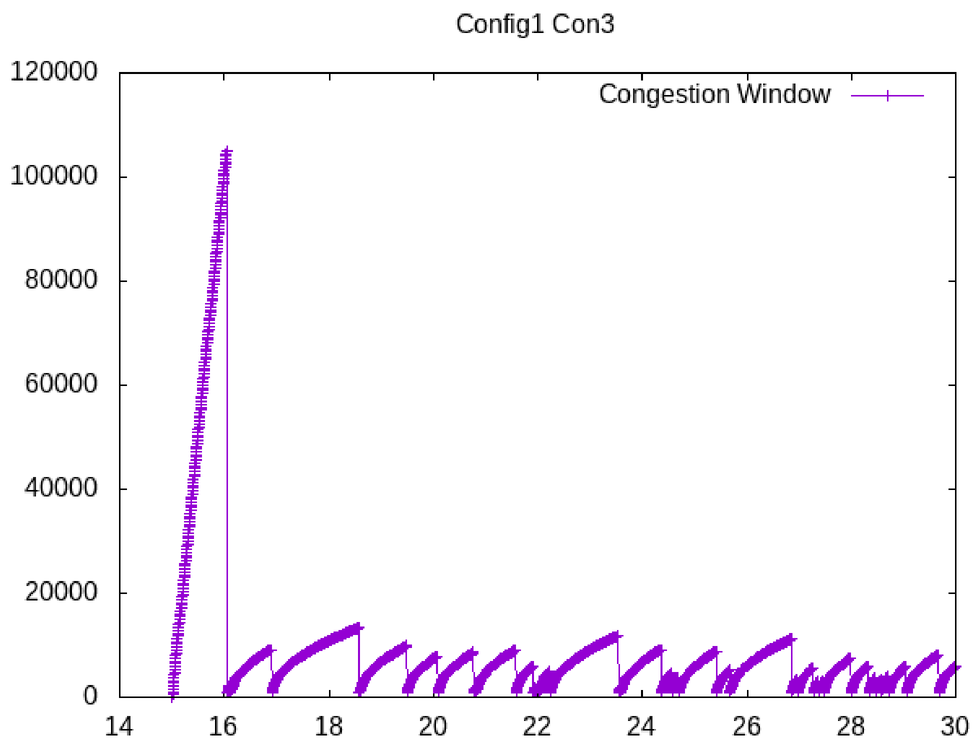
Total number of packets dropped = 113



Here we have plotted the congestion window sizes against time for each connection separately. We can observe that the peak for connection 3 in this configuration is considerably higher than the other 2 connections but as the time proceeds, the values for connection 3 are on the lower side. These observations are due to the fact that connection 1 and connection 2 both work between same source N1 and same sink N3and also because connection 3 starts at a later time.

The number of packets dropped is the highest for this configuration.

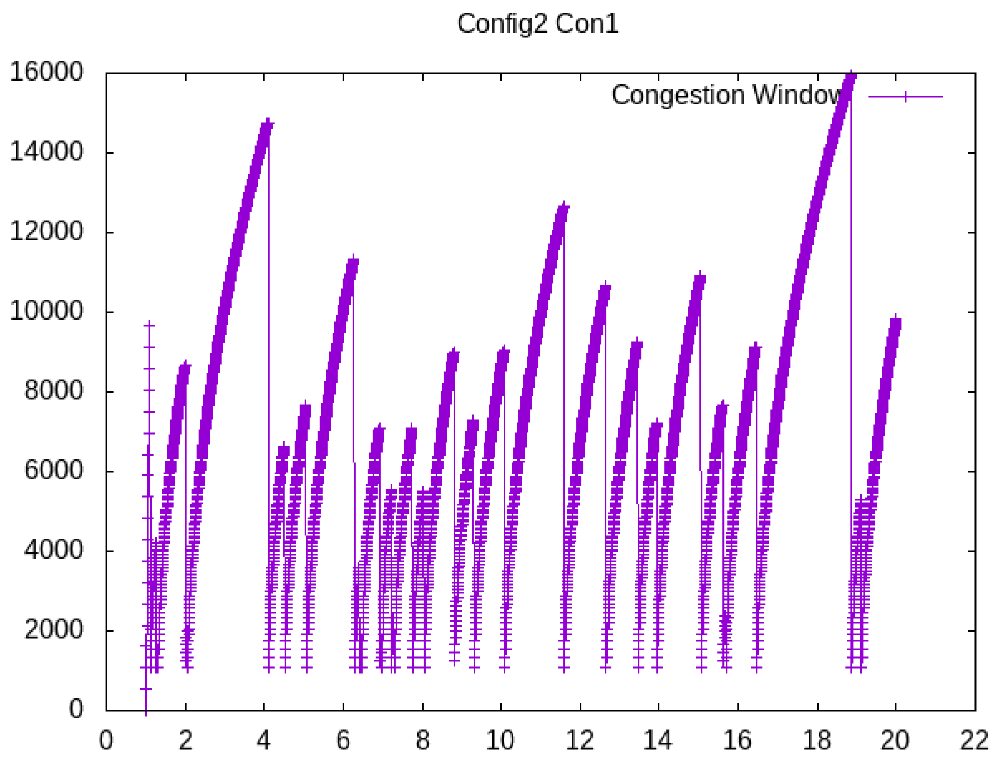

Config1 Con2
Congestion Window
16000
14000
12000
10000
8000
6000
4000
2000
0
4
6
8
10
12
14
16
18
20
22
24
26


Config1 Con3
Congestion Window
120000
100000
80000
60000
40000
20000
0
14
16
18
20
22
24
26
28
30

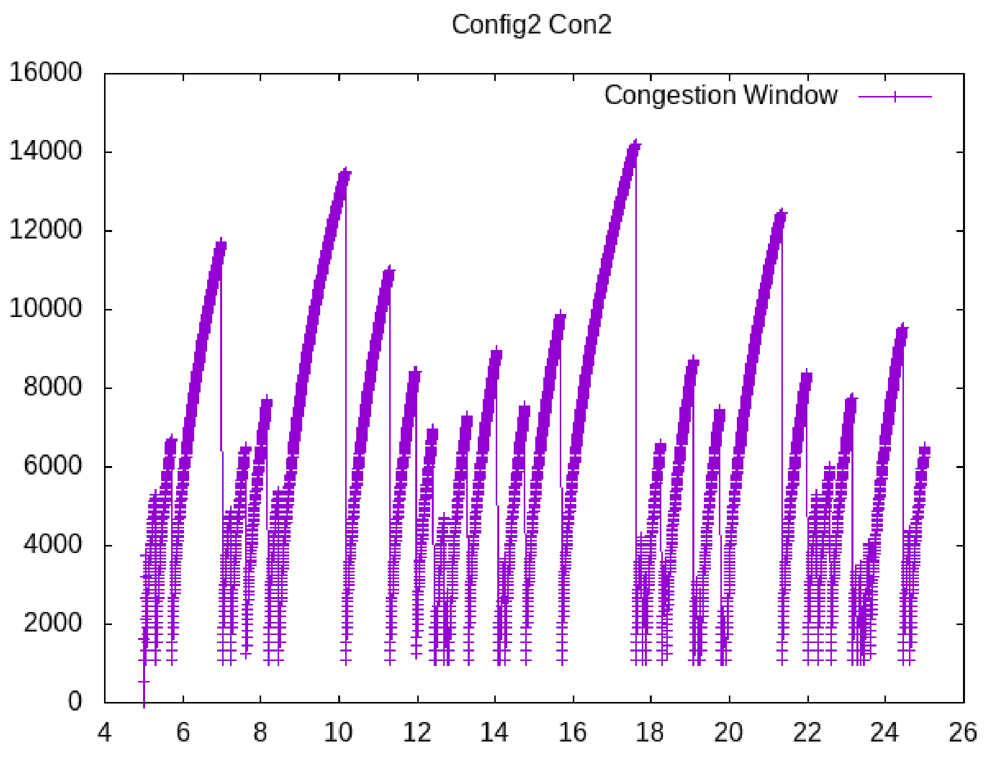
CONFIGURATION 2

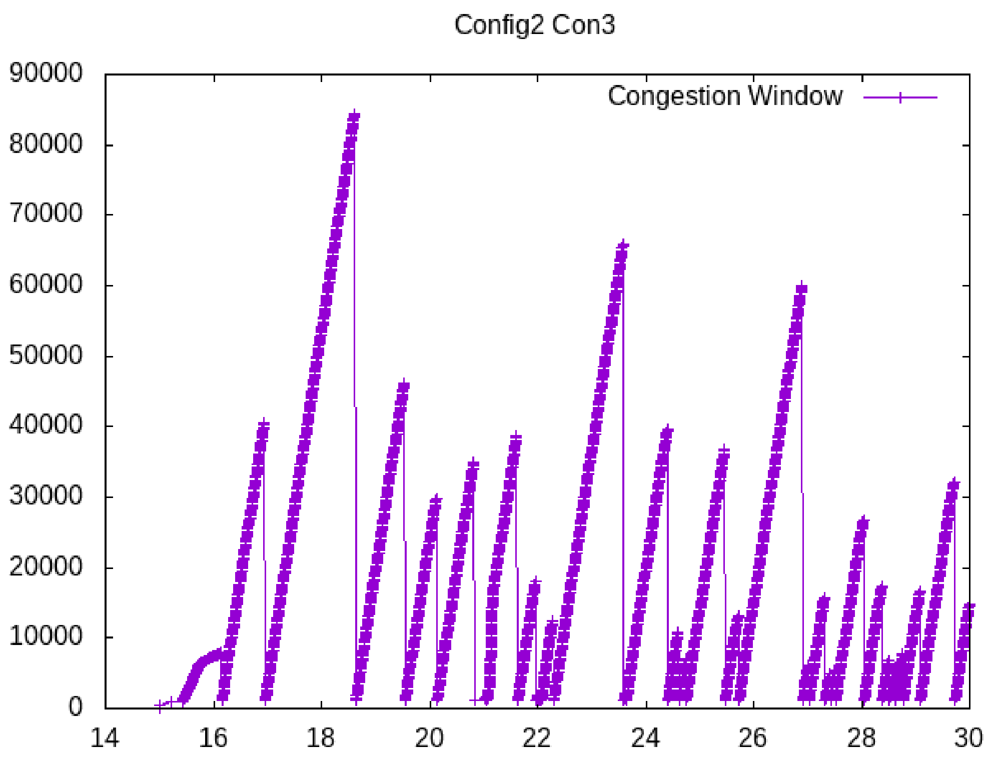
In this configuration, Connection3 at TCP source uses the new congestion protocol TCPNewRenoCSE. The remaining senders use the default protocol TCPNewReno.

Total number of packets dropped = 112



Here we have plotted the congestion window sizes against time for each connection separately . We can observe that the maximum congestion window sizes for connection 3 are considerably higher than the other 2 connections and also the trends look linear. This is due to the fact that connection 3 uses NewRenoCSE at the TCP source which has different algorithm for slow start and congestion avoidance.
The plots for connection 1 and connection 2 are more or less similar, with slight differences probably due to different start times and congestions states.
The number of packets dropped is 1 less than configuration 1.

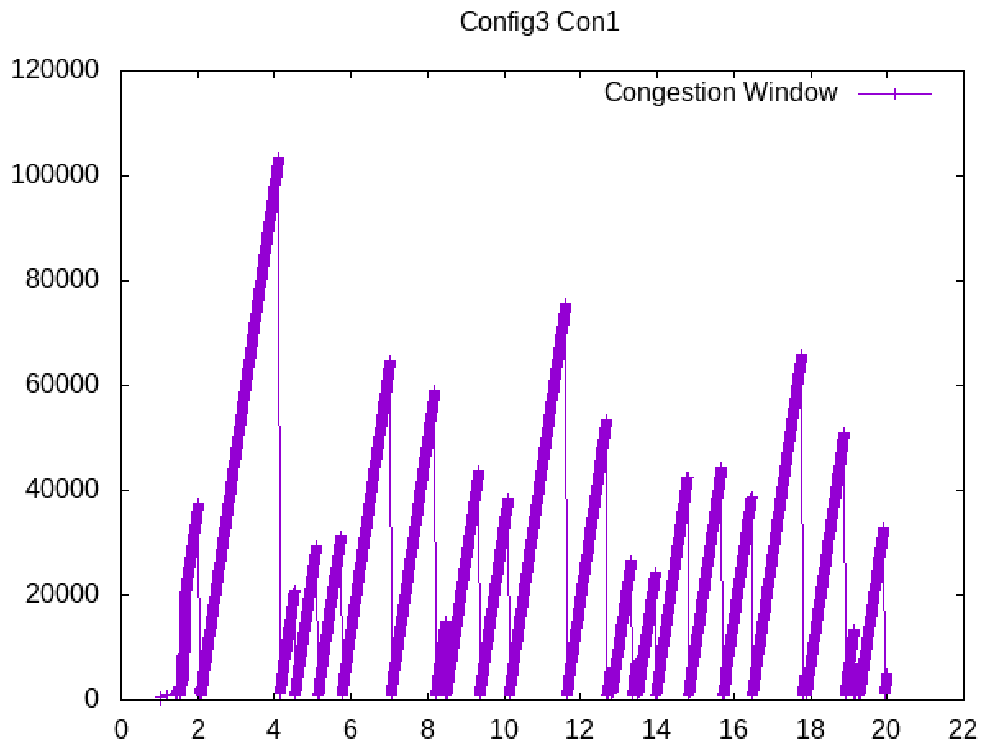

Config2 Con2
Congestion Window
16000
14000
12000
10000
8000
6000
4000
2000
0
4
6
8
10
12
14
16
18
20
22
24
26


Config2 Con3
Congestion Window
90000
80000
70000
60000
50000
40000
30000
20000
10000
0
14
16
18
20
22
24
26
28
30

CONFIGURATION 3

In this configuration, all the senders use TcpNewRenoCSE.

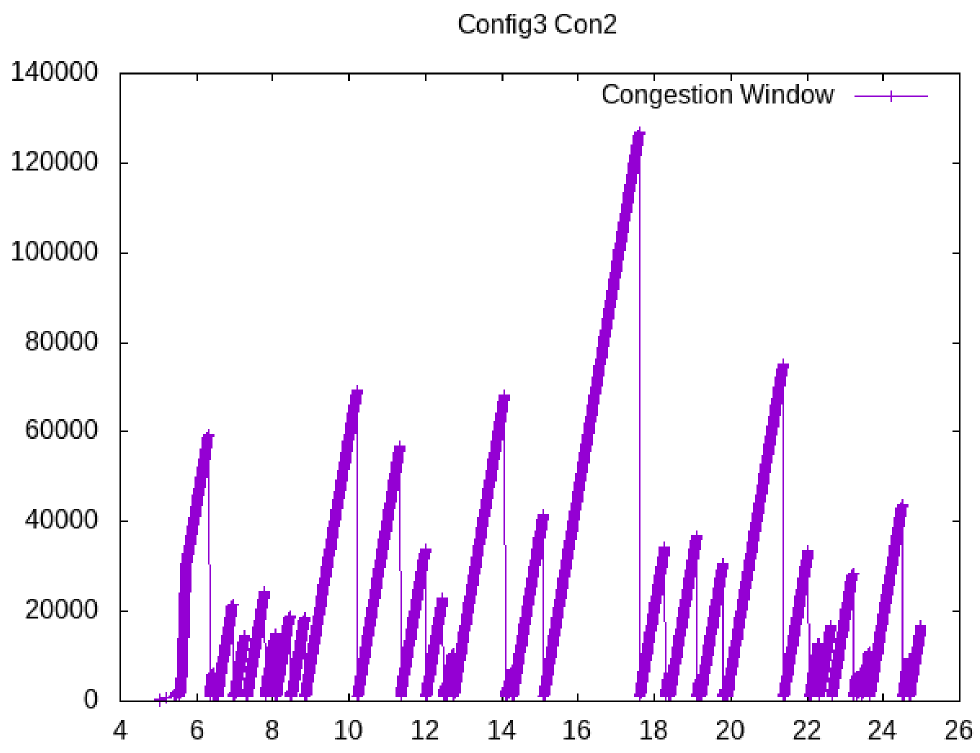
Total number of packets dropped = 110

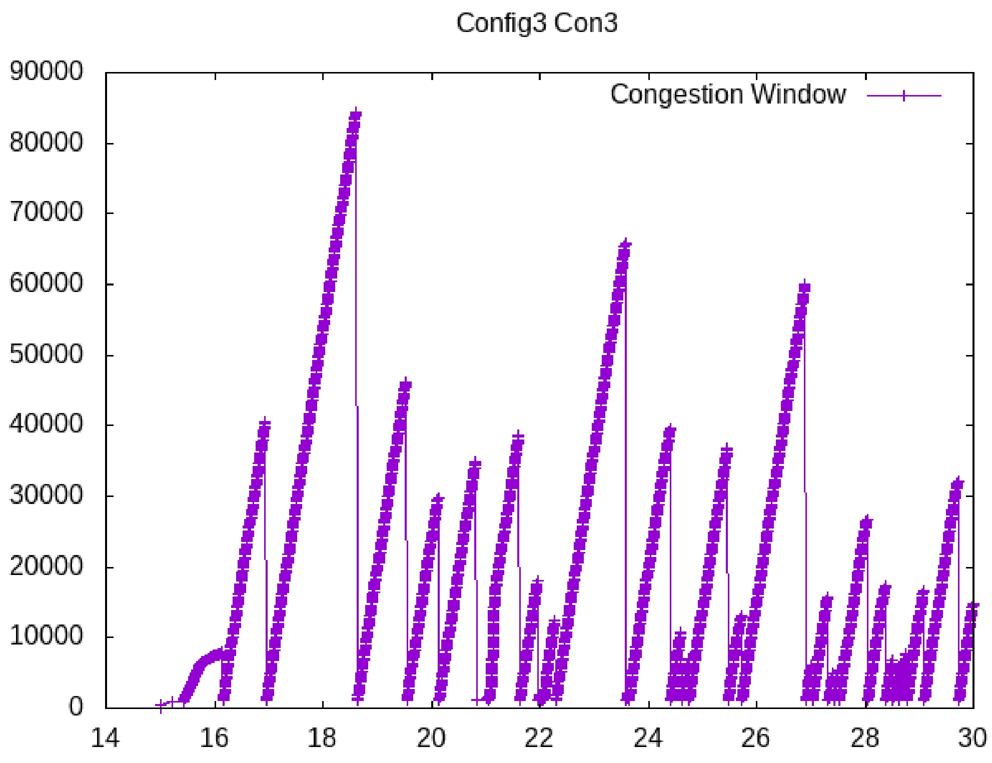


Here we have plotted the congestion window sizes against time for each connection separately . We observe **linear trends** for all the three connections with exceptionally high values for maximum congestion size than the previous configuration due to new algorithm being used for slow start and congestion avoidance by all the sender.

The number of packets dropped is also least among the 3 configurations.

These are the major differences between **NewReno and NewRenoCSE** which has an impact on the entire network as the congestion trends are very different from the above two configurations.


Config3 Con2
Congestion Window
140000
120000
100000
80000
60000
40000
20000
0
4
6
8
10
12
14
16
18
20
22
24
26


Config3 Con3
Congestion Window
90000
80000
70000
60000
50000
40000
30000
20000
10000
0
14
16
18
20
22
24
26
28
30