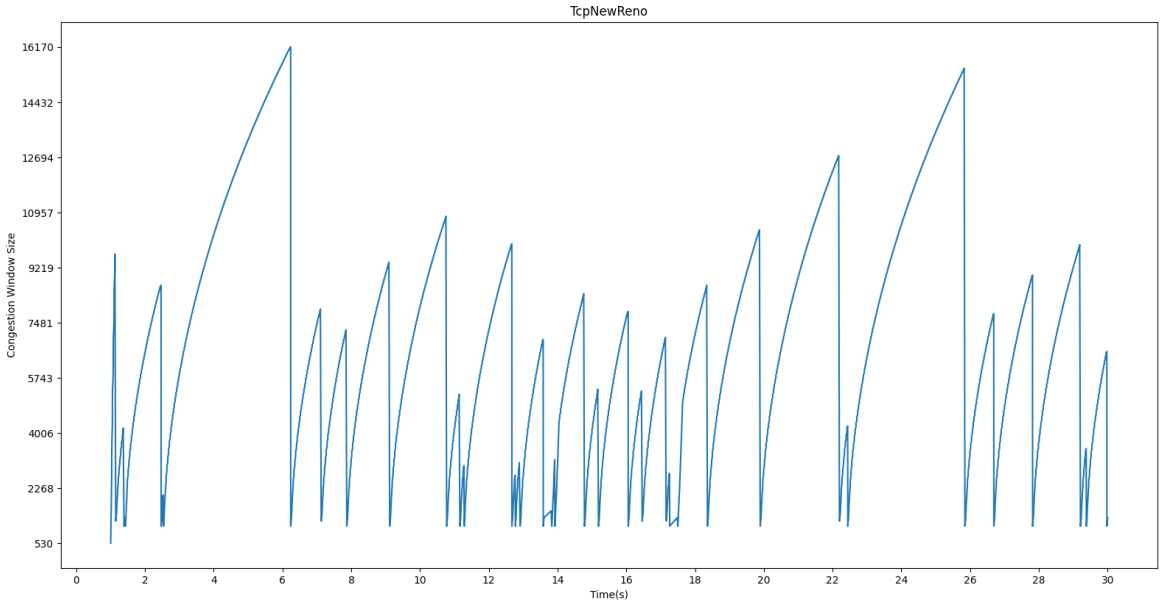
COL 334, Assignment 3

Sarthak Singla, 2019CS10397

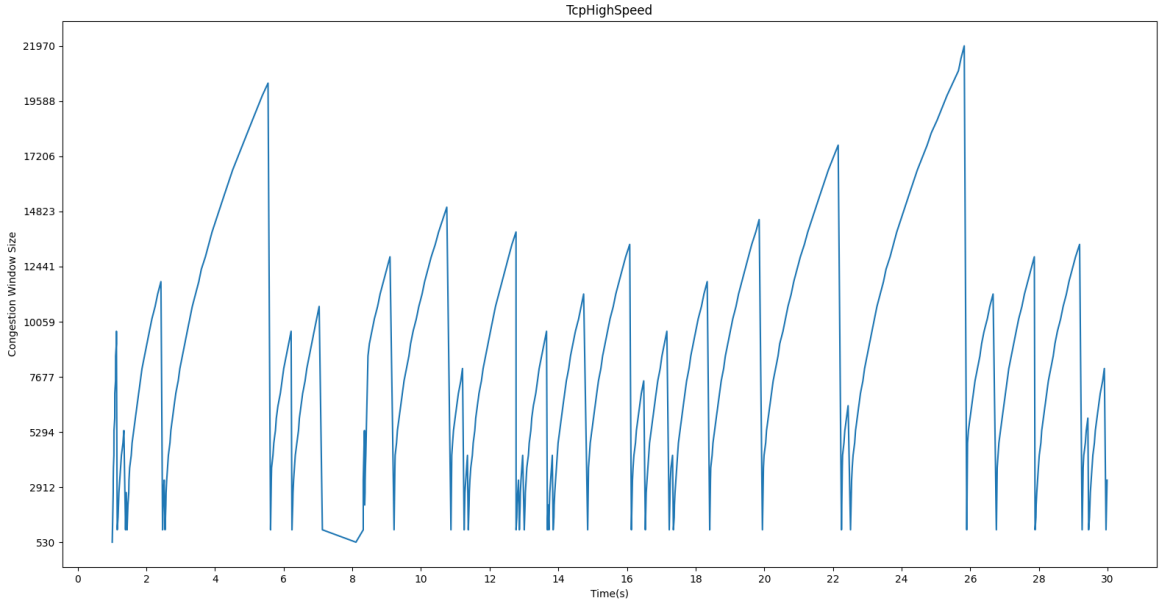
Part1

1. Plots

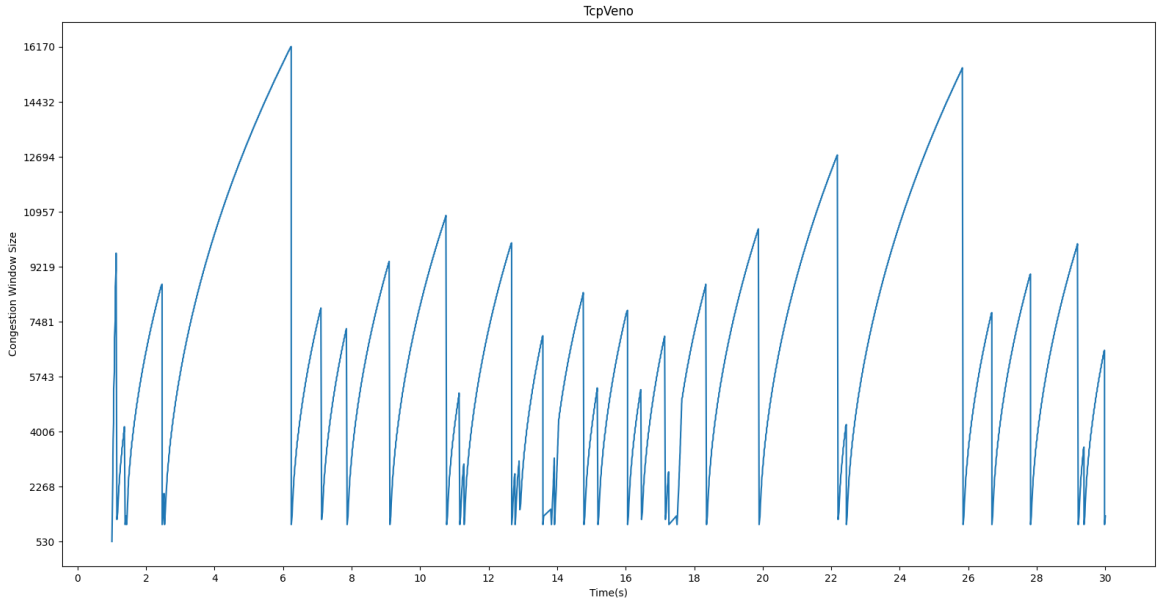
- TcpNewReno



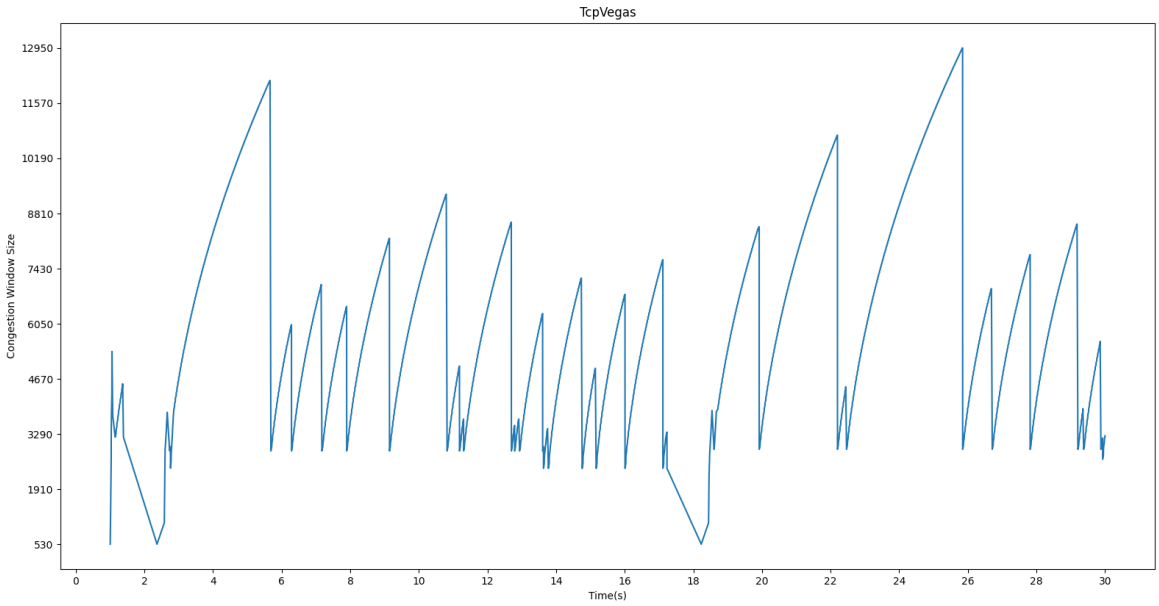
- TcpHighSpeed



- TcpVeno



- TcpVegas



2. Packets dropped

- TcpNewReno - 38
- TcpHighSpeed - 38
- TcpVeno - 38
- TcpVegas - 39

We observe that the number of packets dropped in the three protocols doesn't differ significantly in our example. This is probably because the same rate error model is being used in all the cases which is the cause of packet drop. The slight difference observed for TcpVegas is due to the difference in Cwnd.

3. Observations

- **TCPNewReno**

Here initially in the slow start phase, a linear plot is observed. Here Cwnd <- Cwnd + SegmentSize, which explains the linear plot.

After the first packet drop, concave curves with increase in Cwnd decreasing with Cwnd are observed. Here Cwnd <- Cwnd +(SegmentSize*SegmentSize)/Cwnd, which explains the concave plots.

- **TCPHighSpeed**

It can be observed that TCPHighSpeed leads to higher values of Cwnd compared to the other protocols. The shapes seen however are similar as in TCPNewReno.

The algorithm used in TCPHighSpeed is Cwnd = Cwnd + a(Cwnd)/Cwnd, where a() is provided as a lookup table.

The increase done in TCPHighSpeed is much higher compared to other protocols, which explains us seeing TCPHighSpeed having higher Cwnd compared to other protocols in the graphs. Due to this TCPHighSpeed is suited for and used in high-capacity channels.

- **TCPVeno**

TCPVeno is an enhanced version of TCPReno, which deals more effectively with losses by distinguishing between congested and non-congested states.

It estimates number of packets accumulated at bottleneck queue, N as N = (Cwnd/BaseRTT - Cwnd/RTT)*BaseRTT

After N exceeds some threshold, Veno increases Cwnd only after receiving alternate ACKS, as opposed to after every ACK.

In our plots, we do not see any difference in TCPVeno and TCPNewReno, because the threshold was not reached, and until that the algorithms behave similarly.

- TCPVegas

TCPVegas is a delay-based congestion control algorithm. Therefore, its graph was also very different compared to the above three protocols. Also, as it considers only delays while changing Cwnd, its Cwnd peaks were the lowest compared to other algorithms.

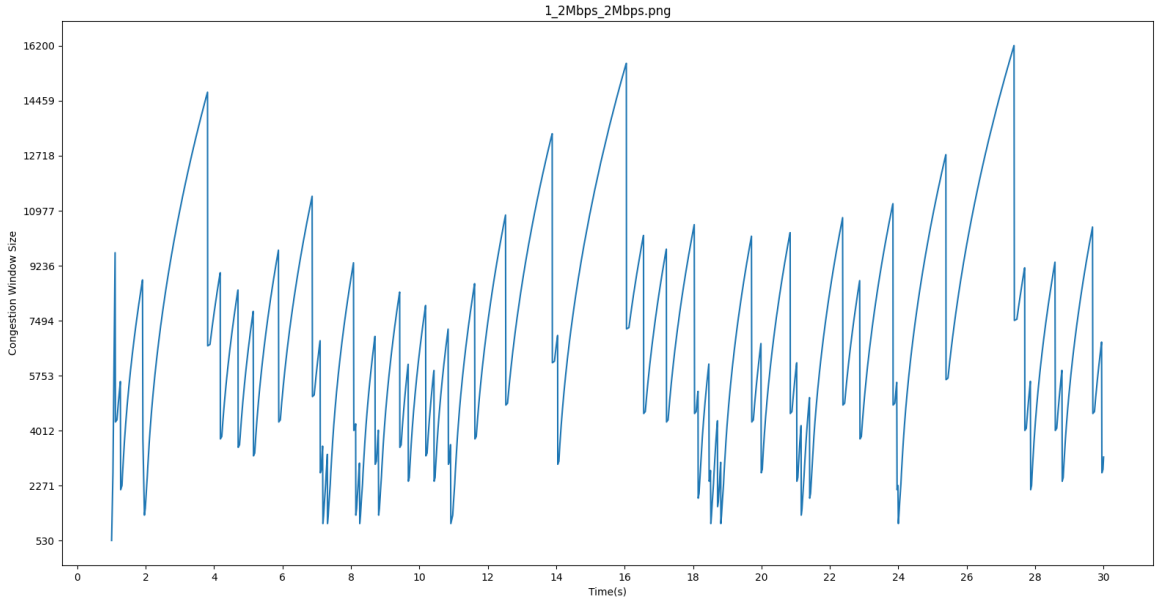
It calcluates:
actual throughput as actual = Cwnd/RTT
expected throughput as expected = Cwnd/BaseRTT
expected-actual estimates number of packets accumulated at bottleneck queue.

TCPVegas tries to maintain the number of accumulated packets between certain thresholds. For this it increases and decreases the congestion window. Due to this, we observe linear decrease in the graph for TCPVegas, which is not present in any other algorithm.

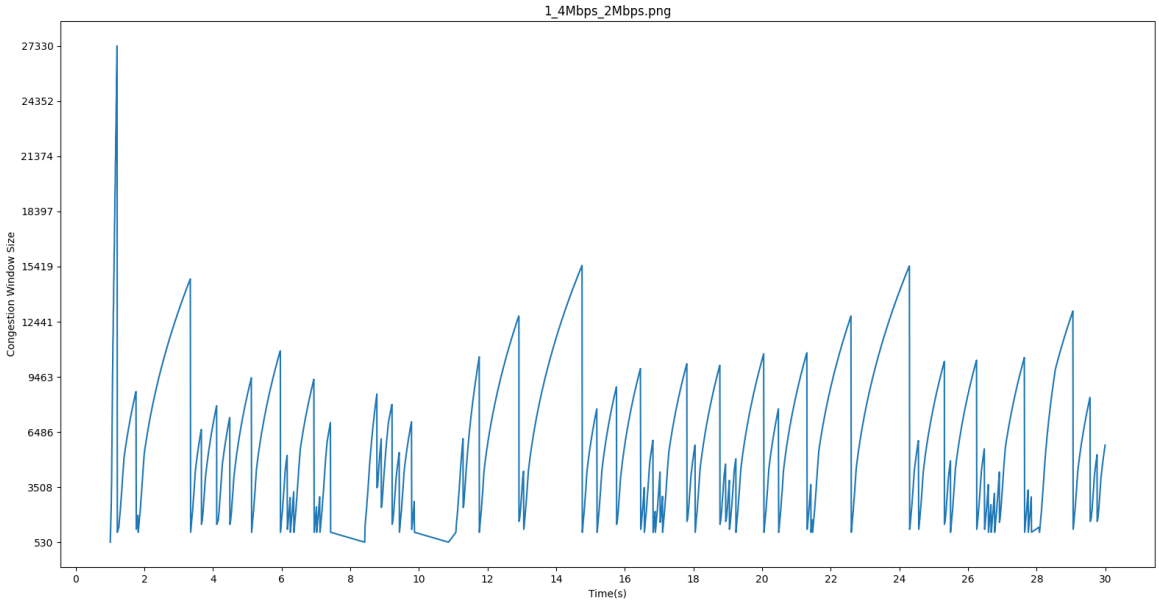
Part 2

(a) Varying Channel Data Rates

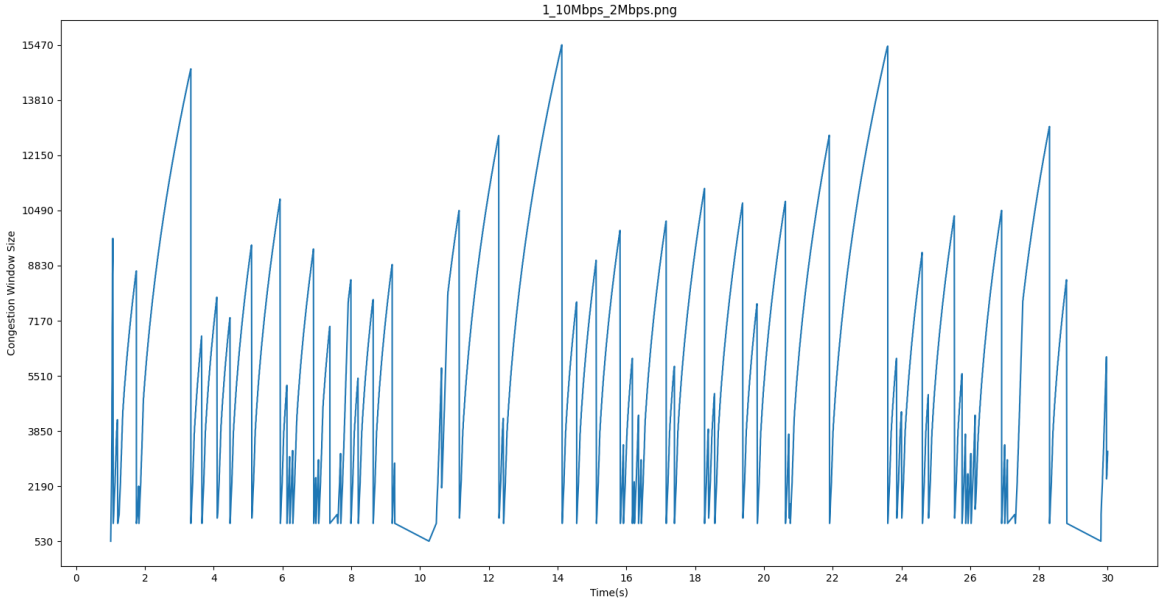
- 2Mbps



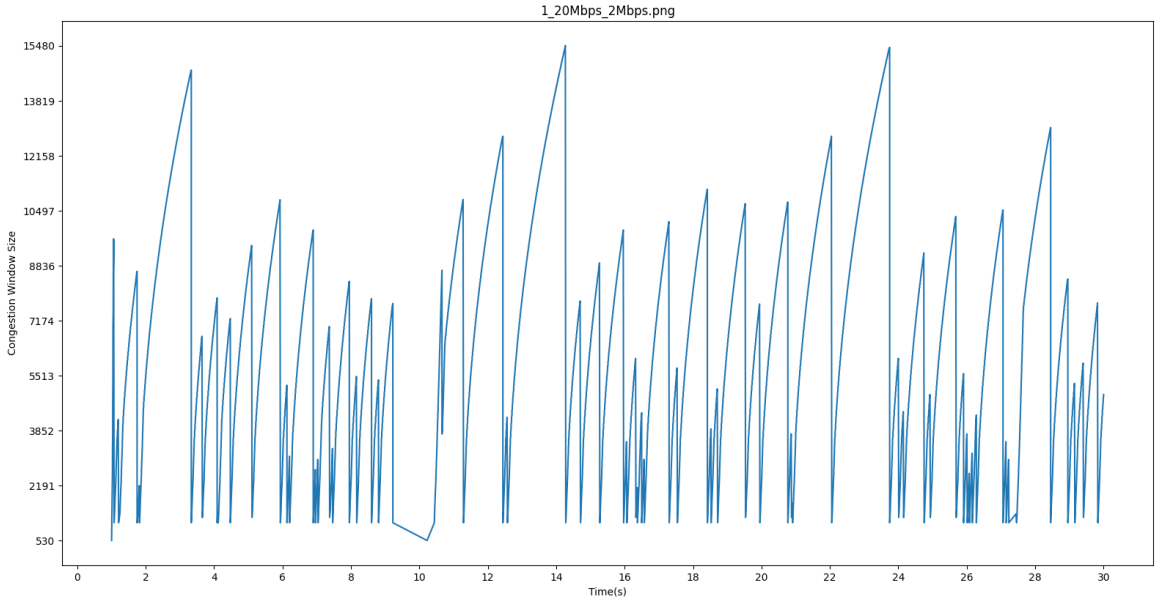
- 4Mbps



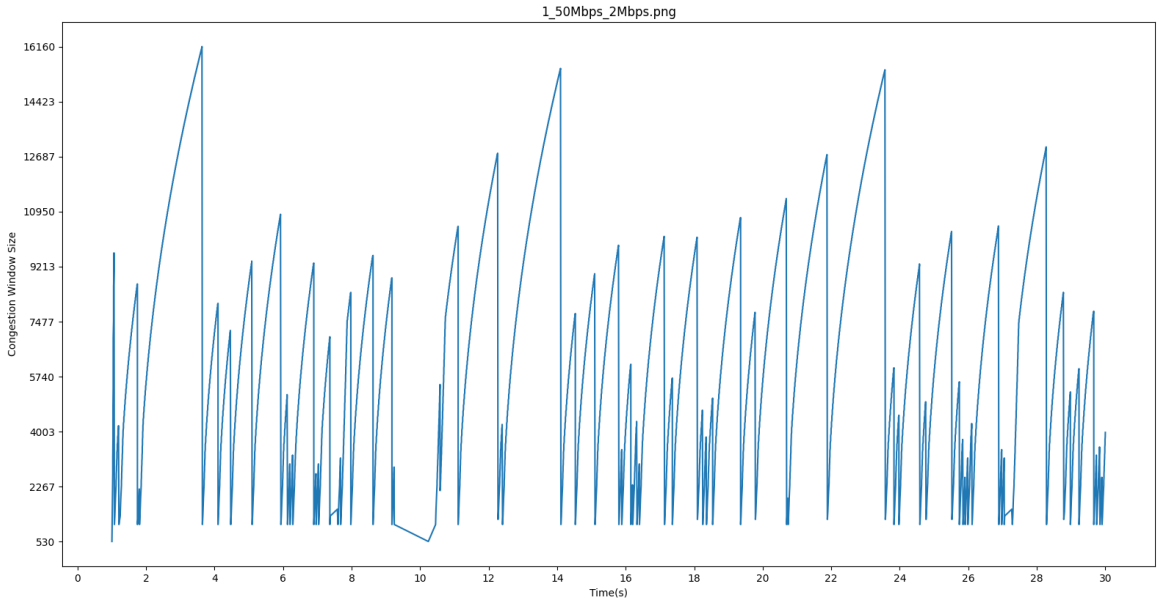
- 10Mbps



- 20Mbps



- 50Mbps



Here the packet drops observed against the Channel Data Rate against fixed application Data Rate of 2Mbps were:
2Mbps - 62
4Mbps - 72
10Mbps - 73
20Mbps - 74
50Mbps - 75

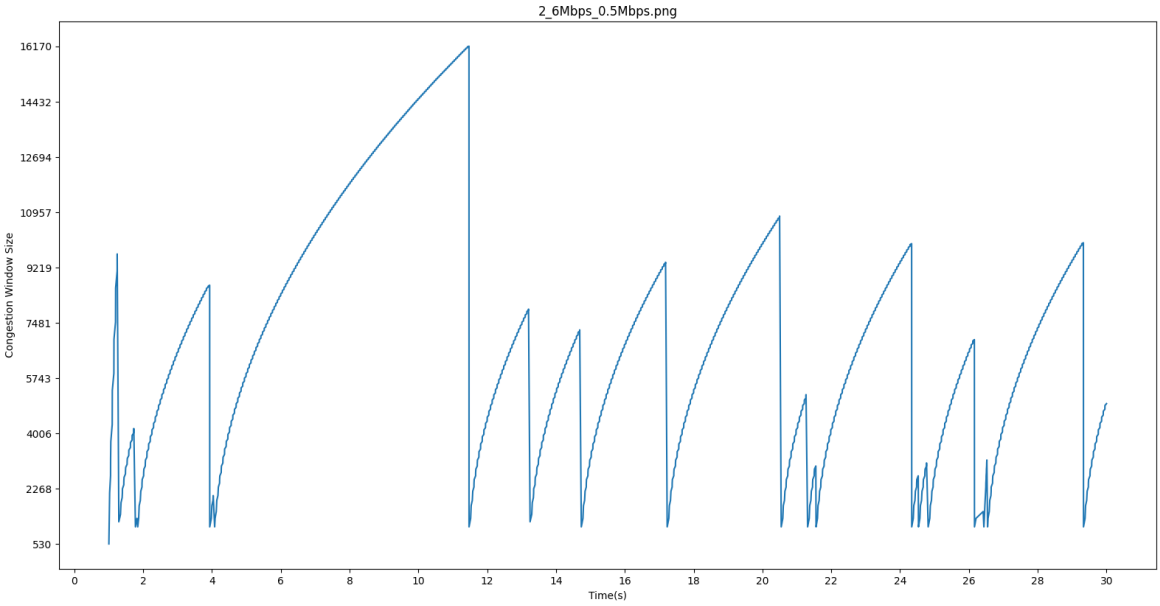
There is a significant increase only going from 2 to 4Mbps. This shows that the packet drops do not increase significantly after the Channel Data Rate is larger than the Application Data Rate.

Also, we can see the peaks do not change significantly over the various values of Channel Data Rate.

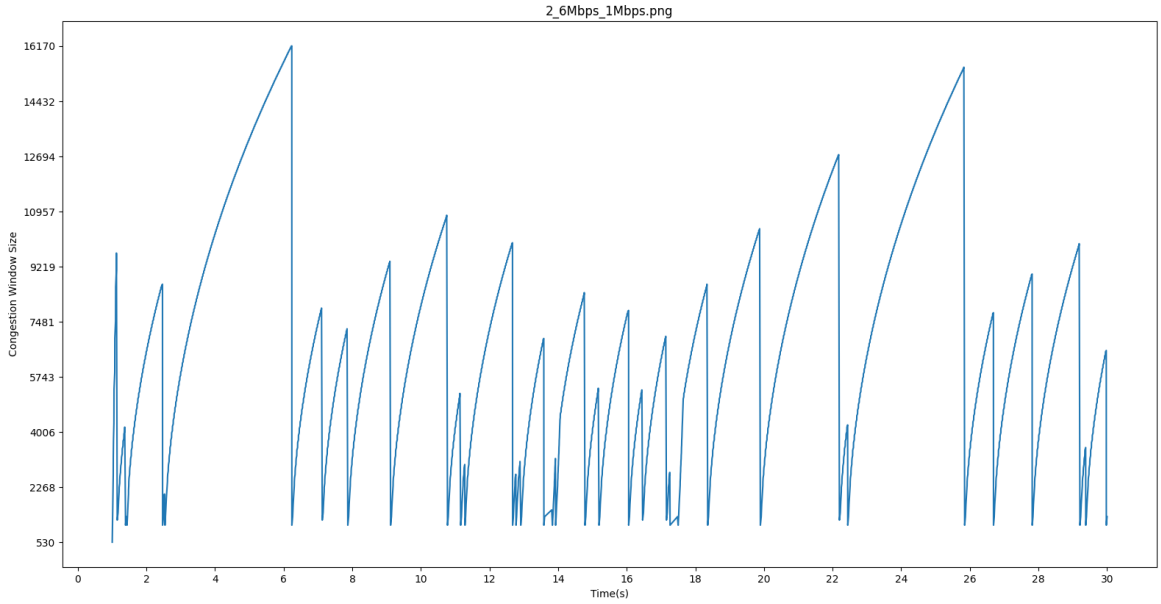
These are expected because when the Channel Data Rate is more than Application Data Rate, there is no change in application induced congestion because only a single link is present and channel already allows transmission at higher rate than application wants.

(b)Varying Application Data Rates

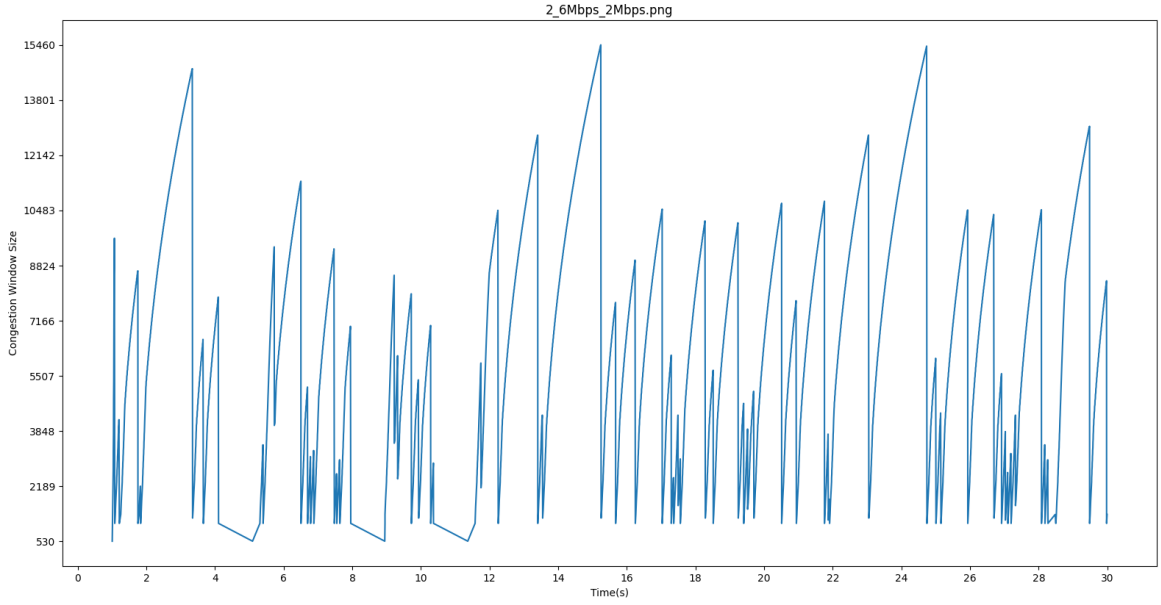
- 0.5Mbps



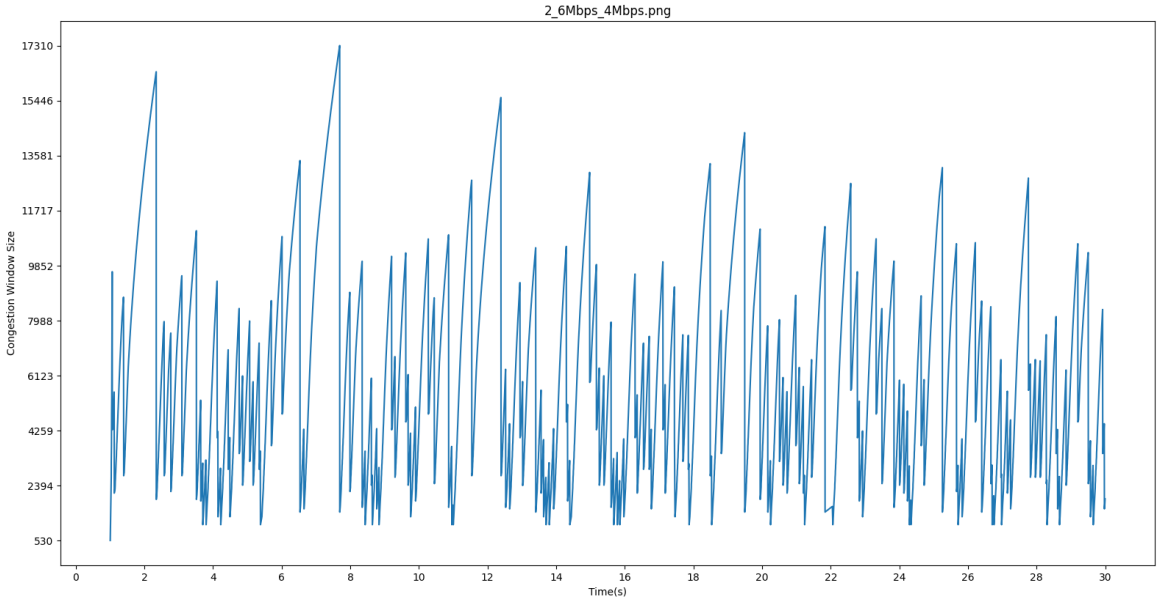
- 1Mbps


2_6Mbps_1Mbps.png
Congestion Window Size
16170
14432
12694
10957
9219
7481
5743
4006
2268
530
0
2
4
6
8
10
12
14
16
18
20
22
24
26
28
30
Time(s)

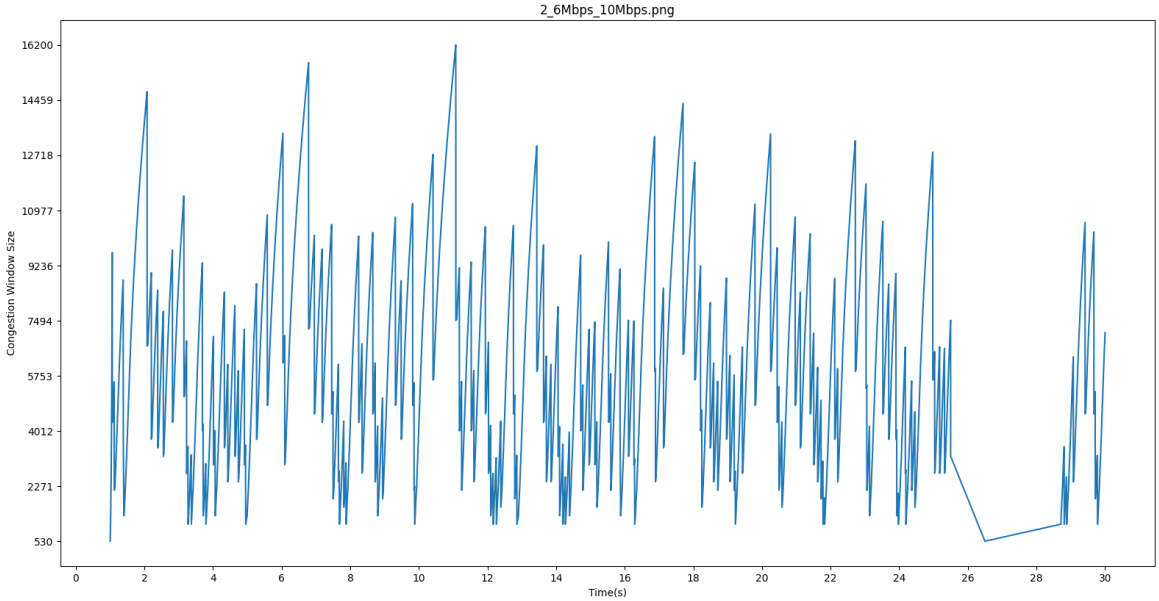
- 2Mbps


2_6Mbps_2Mbps.png
Congestion Window Size
15460
13801
12142
10483
8824
7166
5507
3848
2189
530
0
2
4
6
8
10
12
14
16
18
20
22
24
26
28
30
Time(s)

- 4Mbps



- 10Mbps



The packet drops observed for varying Application Data Rates against fixed Channel Data Rate of 6Mbps were:

0.5Mbps - 22
1Mbps - 38
2Mbps - 71
4Mbps - 156
10Mbps - 156

Here we can clearly observe that increasing the application data rate increases the number of packets dropped. Also, after a certain limit, the number of packets dropped doesn't increase as observed from the 4Mbps and 10Mbps example. This is because of the channel data rate limiting.

Here it is clearly observable that the plots become congested with an increase in the application data rate. This is because, initially, the application data rate is lower than the channel data rate, so the application can increase transmission rate and channel has capacity for that. Also, due to increased transmission, we see compression of plots.

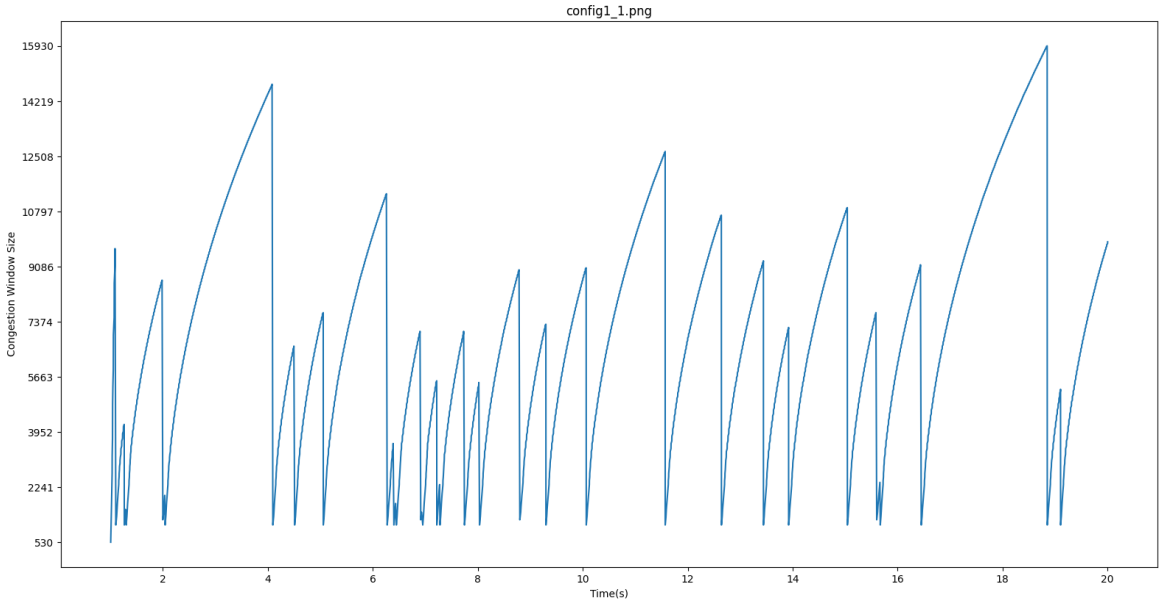
Part 3

- I have taken two separate error models at N3 for the two links.

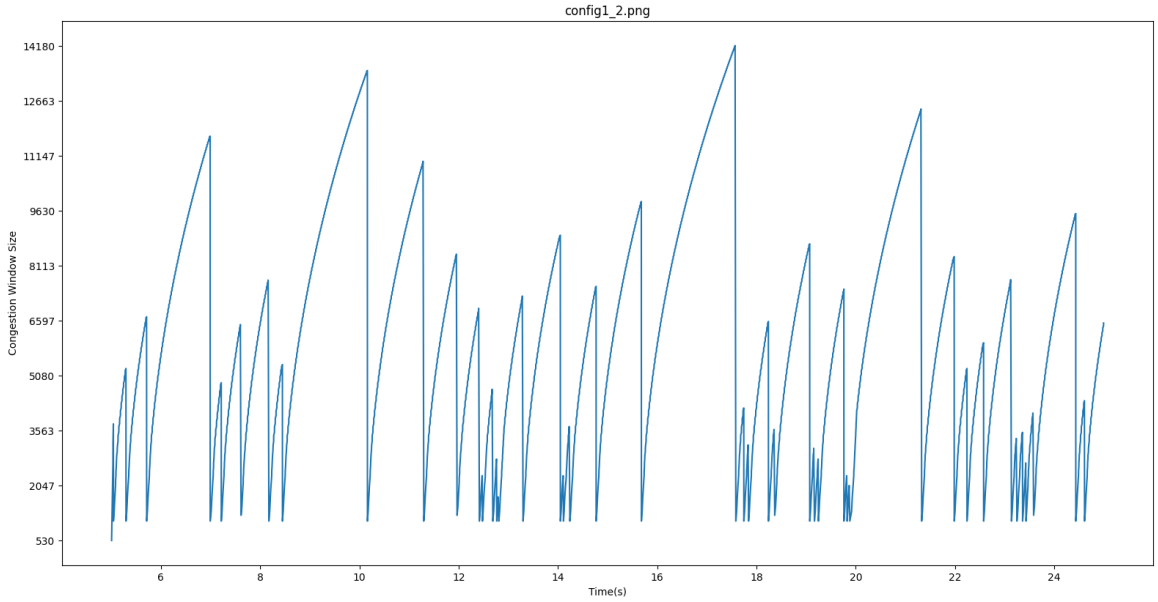
Configuration 1

1. Plots

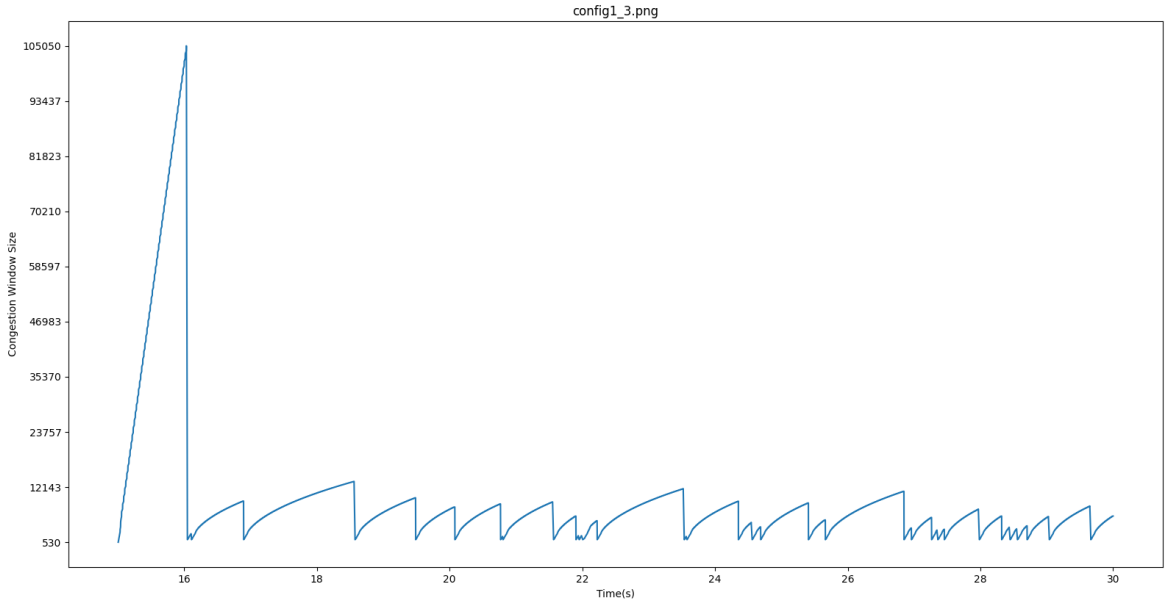
- Connection 1



- Connection 2



- Connection 3



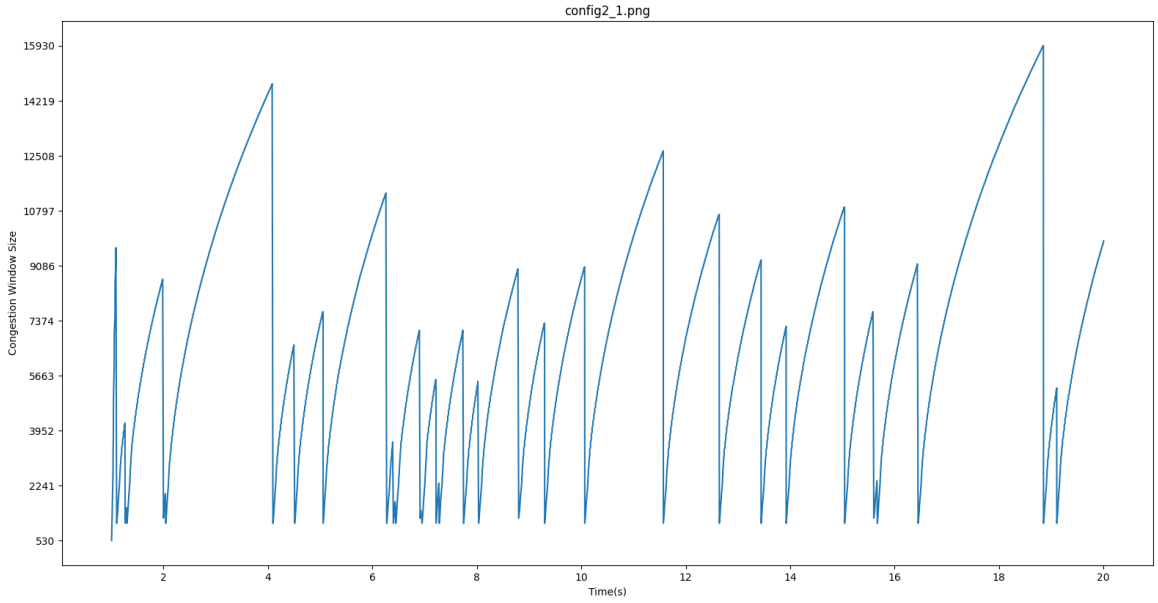
2. Packet Drops

The total number of dropped packets over all connections was 113.

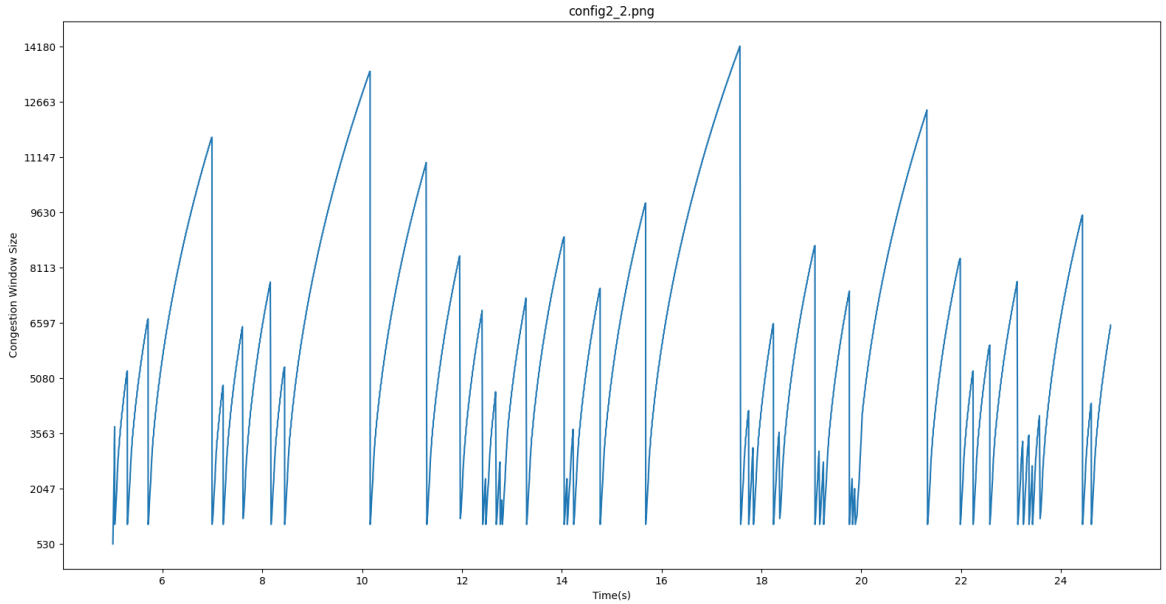
Configuration 2

1. Plots

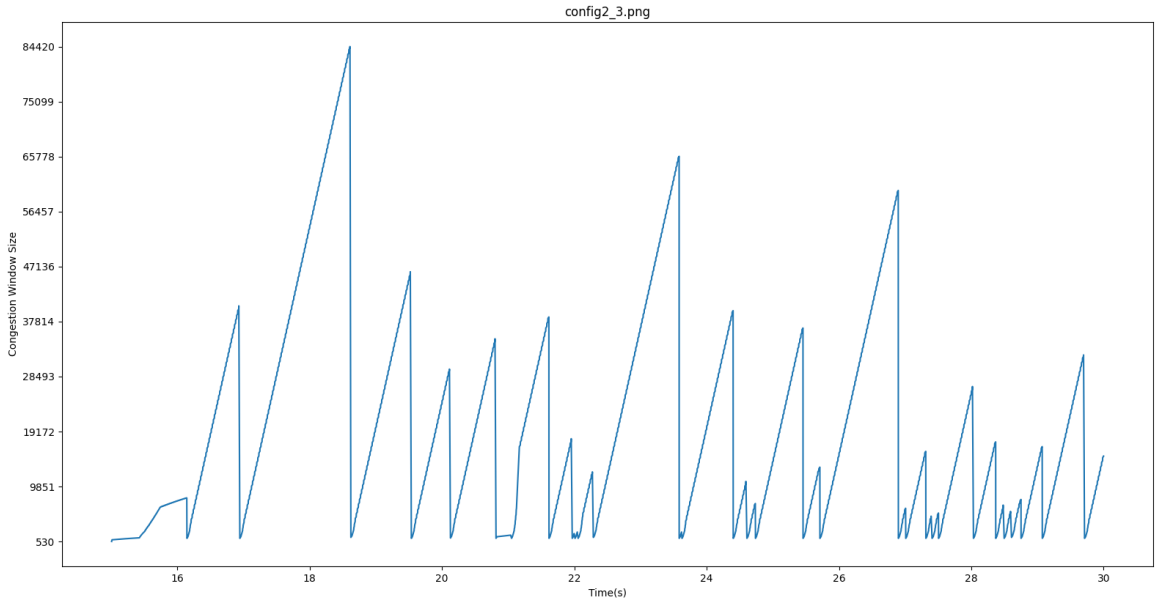
- Connection 1



- Connection 2



- Connection 3



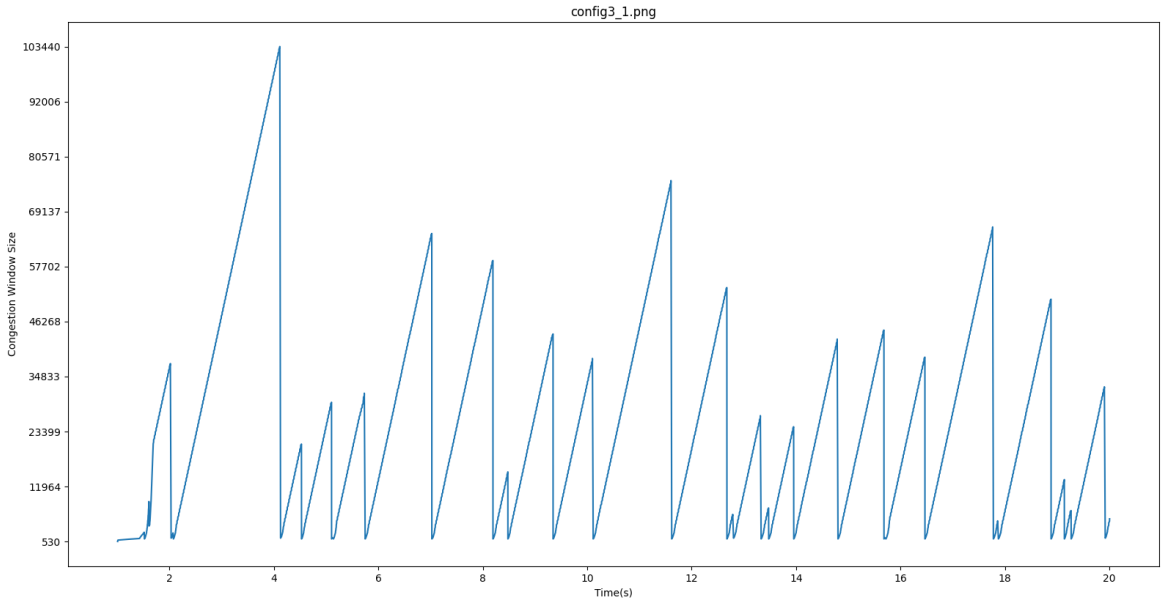
2. Packet Drops

The total number of dropped packets over all connections was 112.

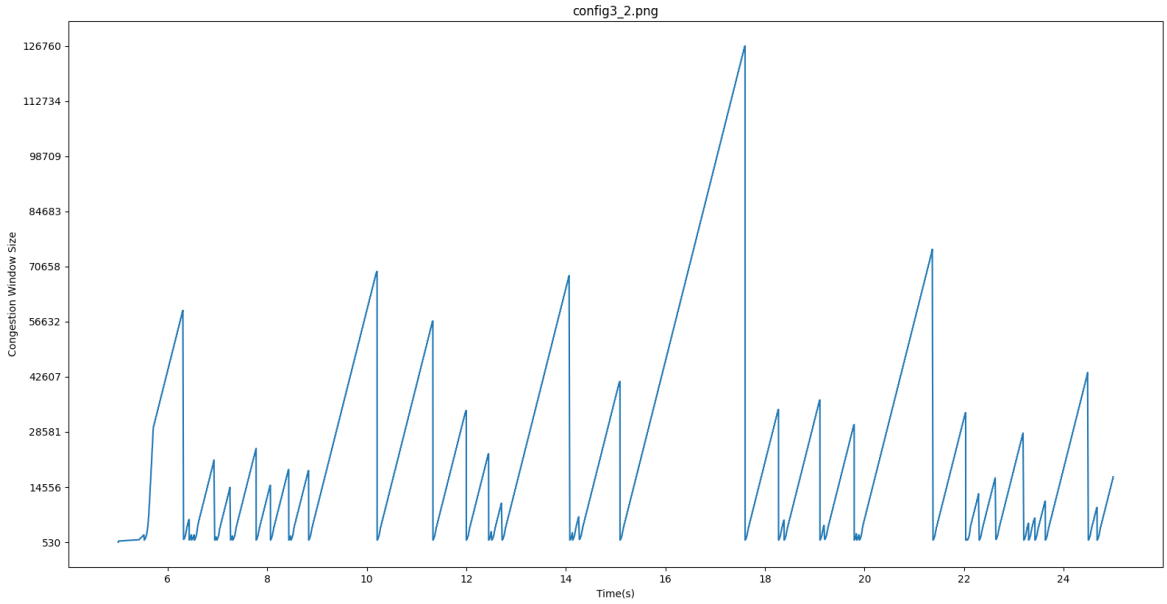
Configuration 3

1. Plots

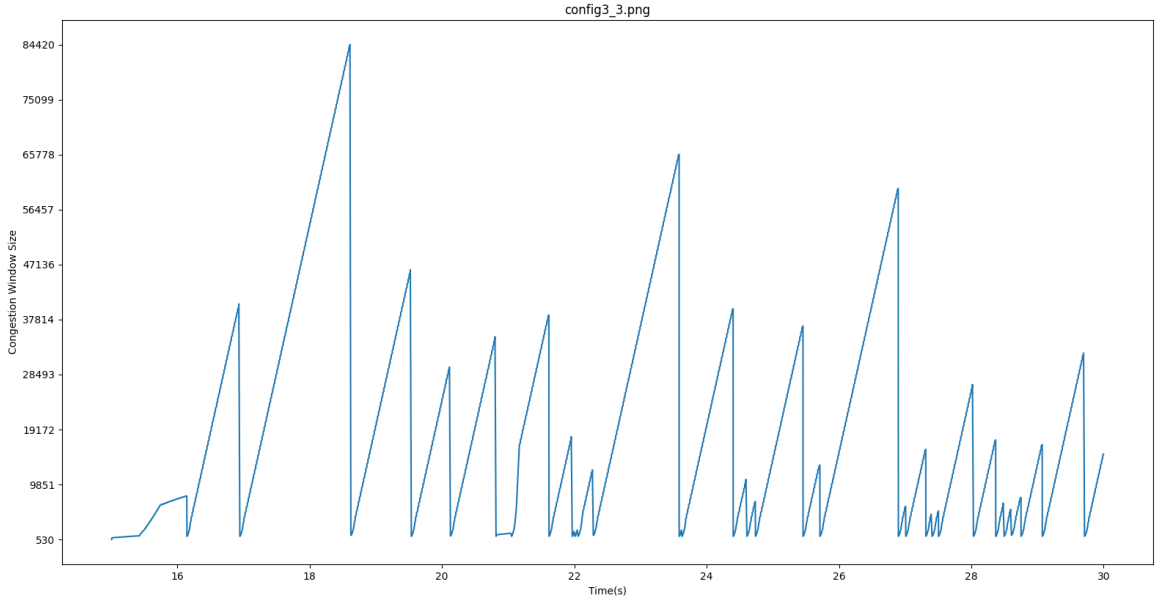
- Connection 1



- Connection 2



- Connection 3



2. Packet Drops

The total number of dropped packets over all connections was 110.

3. Conclusion

How does the congestion avoidance phase vary on the same sender when using TCPNewRenoCSE vs TCPNewReno

For this, we observe the difference between plots of connection 3 in configurations 1 and 2. In configuration 1, connection 3 uses TCPNewReno, while in configuration 2 it uses TCPNewRenoCSE. Both the other connections are kept constant, which helps us to contrast.

We observe that in the congestion avoidance phase, in TCPNewReno, the increase decreases with an increase in the size of the congestion window, and we observe some saturation of the congestion window size with an increase in it. However in TCPNewRenoCSE, the increase in the size of the congestion window is constant, and the congestion window shows a linear increase in the congestion avoidance phase. This is observed from the concave curved shape of the congestion window in case of TCPNewReno while a linear shape in case of TCPNewRenoCSE.

Similar shapes of curves can also be contrasted for connection 1 with configuration 2 and 3 and connection 2 with configuration 2 and 3, as the protocol used by the sender changes.

Apart from this trend in shape, it is also observed that the congestion window size remains at much larger values in case of TCPNewRenoCSE compared to TCPNewReno. For example with TCPNewReno, in congestion avoidance phase connection 3 has congestion window size not exceeding values around 13,000. But for TCPNewRenoCSE the congestion window sizes go upto 30-70 thousand.

Difference in algorithms -

In the congestion avoidance phase of TCPNewReno the congestion window increases by (SegmentSize*SegmentSize)/Cwnd. In the congestion avoidance phase TCPNewRenoCSE, the congestion window increases by 0.5*SegmentSize. This explains the concave curve observed for TCPNewReno and the linear curve for TCPNewRenoCSE.

How does it impact the entire network?

We see that changing connection3 to TCPNewRenoCSE does not affect connection 1 and 2, as plots of connections 1 and 2 in configurations 1 and 2 are same. This is because these connections use two different links.

TCPNewReno tries to slow down increase of Cwnd in Congestion Aviodance phase that makes Cwnd for it lower than Cwnd for TCPNewRenoCSE.

On changing all the senders to TCPNewRenoCSE, we observed that the sizes of congestion windows increased a lot. Therefore, TCPNewRenoCSE is suited in channels where high capacity is available, so we would get higher congestion window sizes.

However, TCPNewRenoCSE has a problem. Its slow start phase tries to keep the increase of Cwnd very low. Due to this, until an error occurs, the Cwnd size remains very low.

Overall, we also observe that in our current scenario TCPNewRenoCSE over all the senders has less packet loss than TCPNewReno over all senders, but the change is marginal and may not have a direct cause.

Description of submitted files

- First.cc - This is the file for Part1. It runs the simulation of TCP connection between two nodes. Different protocols can be run via the argument --TcpVersion=<TcpProtocol>
- Second.cc - This is the file for Part2. Different Channel Data Rates can be used with fixed Application Data Rate of 2Mbps using argument --CDataRate=<Channel Data Rate>. Different Application Data Rates can be used with fixed Channel Data Rate of 6Mbps using argument --ADataRate=<Application Data Rate>.
  Here Mbps must be used while specifying the Data Rates.
- Third.cc - This is the file for Part3. Different Configurations according to question can used using the argument --Config=<Configuration>. Here configuration are 1,2,3
- TCPNewRenoCSE.cc and TCPNewRenoCSE.h - These are the files containing the new protocol TCPNewRenoCSE. This must be kept in src/internet/models.

For above files to compile, they must be mentioned in appropriate wscripts.

plotter_first.py - this file is used to generate plots for Part 1. <TcpVersion> must be specified as command line argument.

plotter_second.py - this file is used to generate plots for Part 2. 2 command line arguments are required. For plots of subpart (a), mention 1 and CDataRate. For plots of subpart (b), mention 2 and ADataRate.

plotter_third.py - this file is used to generate plots for Part 3. Mention 1, 2 or 3 as the command line argument for plots of configuration 1, 2 or 3 respectively.

Above files must be kept in the directory where output of First.cc, Second.cc and Third.cc respectively are stored.