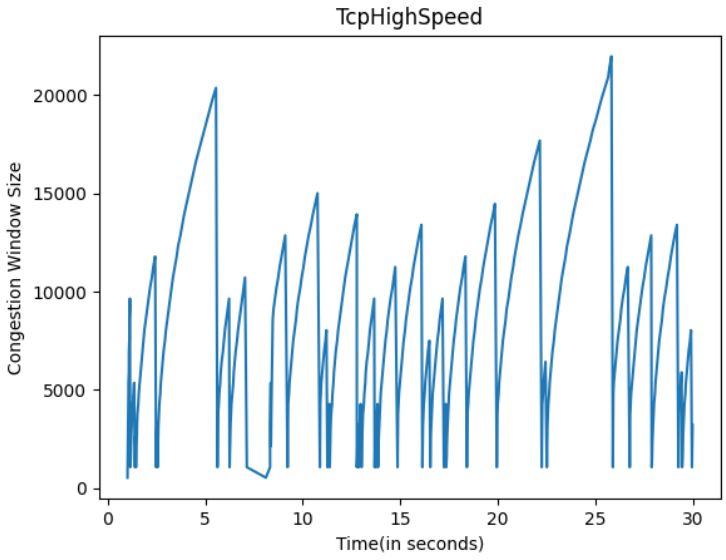
COL334 Assignment 3

- by Dhairya Gupta, 2019CS50428

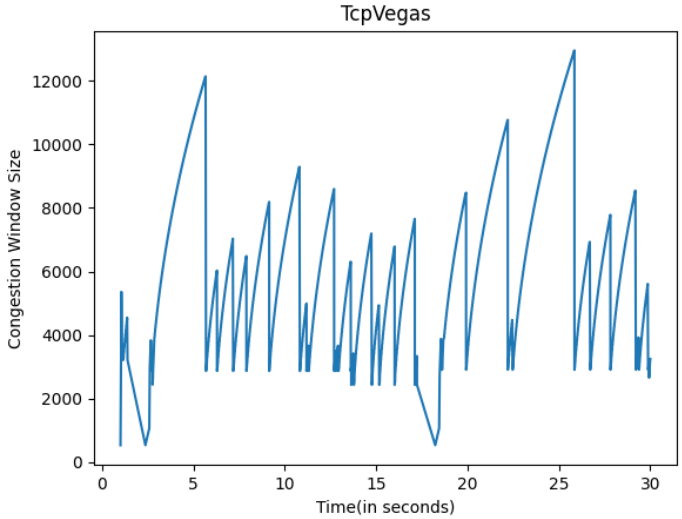
Q1)

1. Plot for TcpHighSpeed (No. Of dropped packets = 38.)

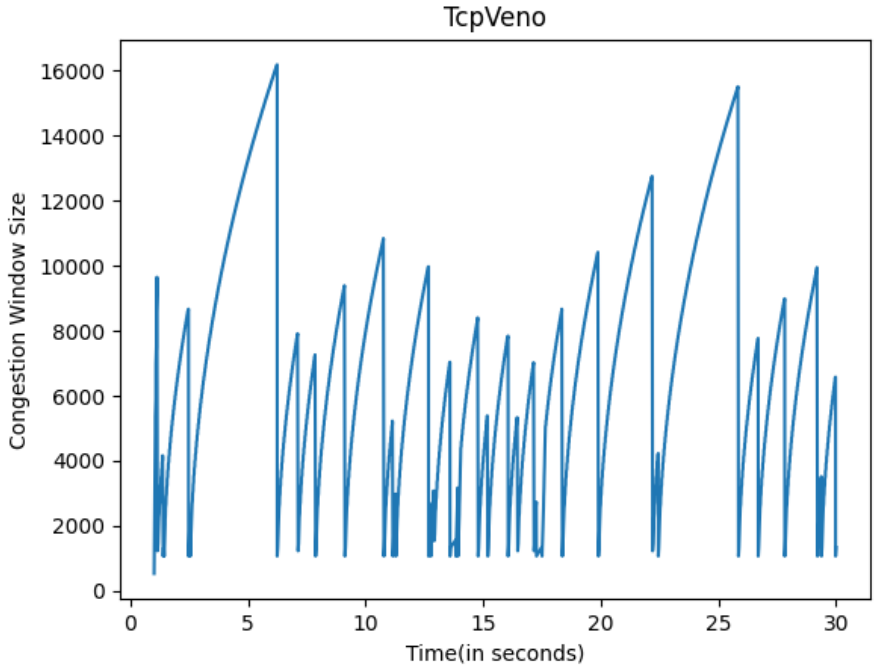
‘



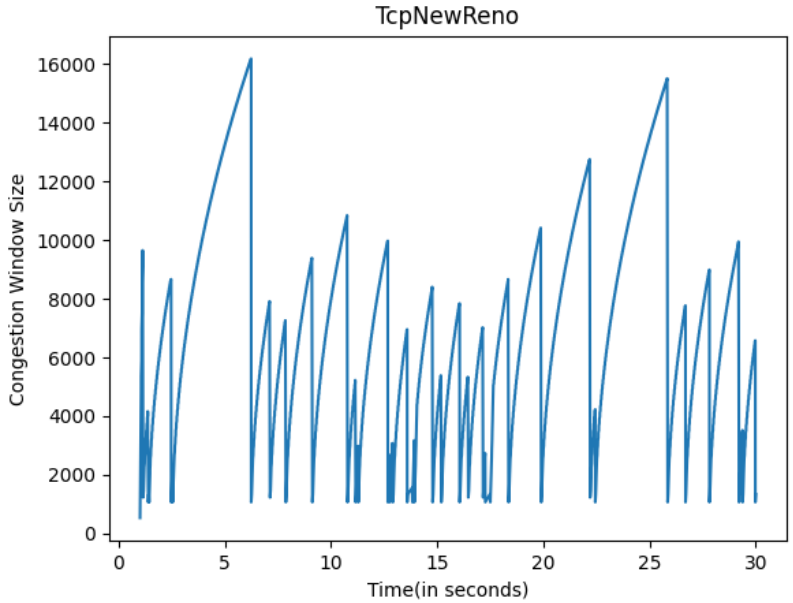
2. Plot for TcpVegas (No. Of dropped packets = 39)



3. Plot for TcpVeno(No. Of dropped packets = 38)



4. Plot for TcpNewReno(No. Of dropped packets = 38)



Observations:

(a) TcpNewReno:

From the graph of TcpNewReno, we observe that the congestion window size(cwnd) drops to a non-zero value(approx. 1072) when a packet drop oocurs. The maximum congestion window size ~ 16000. In congestion avoidance phase, cwnd goes to cwnd + MSS(max Segment Size).
In the slow start phase, congestion window increases by min(Unacknowledged bytes, MSS).
Goes into fast-retransmit mode if multiple duplicate ACKs are detected.

(b) TcpHighSpeed:

Comgestion window drops similar to TcpNewReno(1072 bytes).
The slow start phase (concave portion of graph) is followed by linear congestion avoidance phase.
Maximum Congestion Window size is ~20000 bytes.
Observed jumps in graph are higher than other variants.
TcpHighSpeed is more suitable to be used in channels with high bandwidth-delay product than standard TCP. Average window size is higher in TcpHighSpeed.

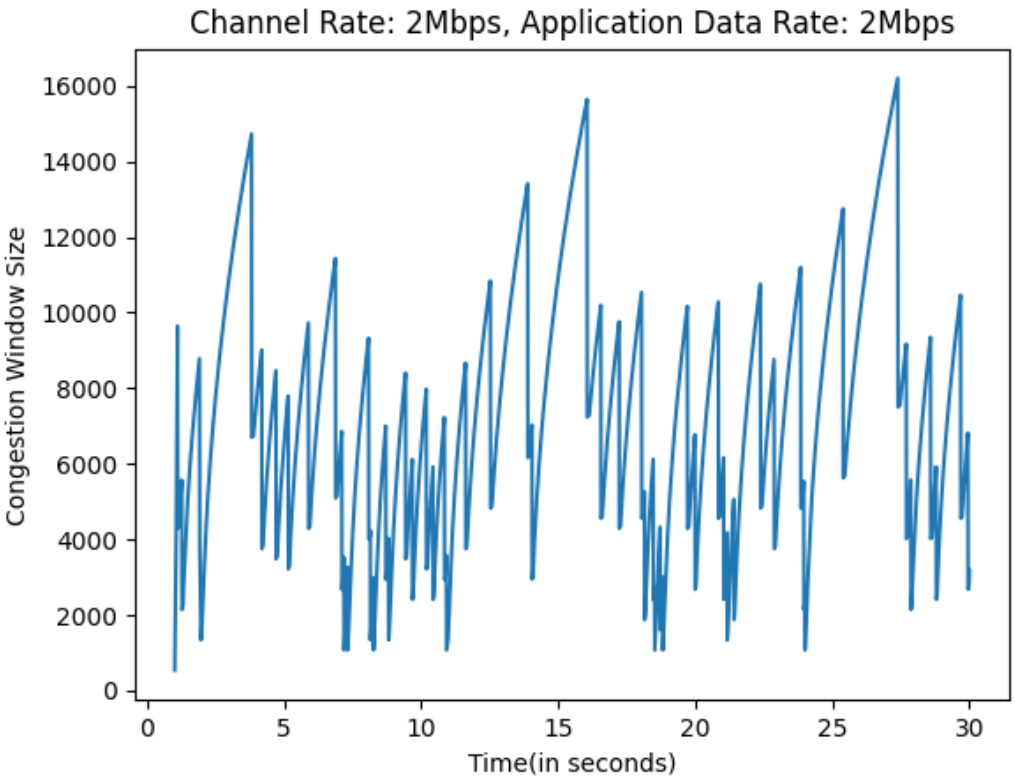
(c) TcpVegas

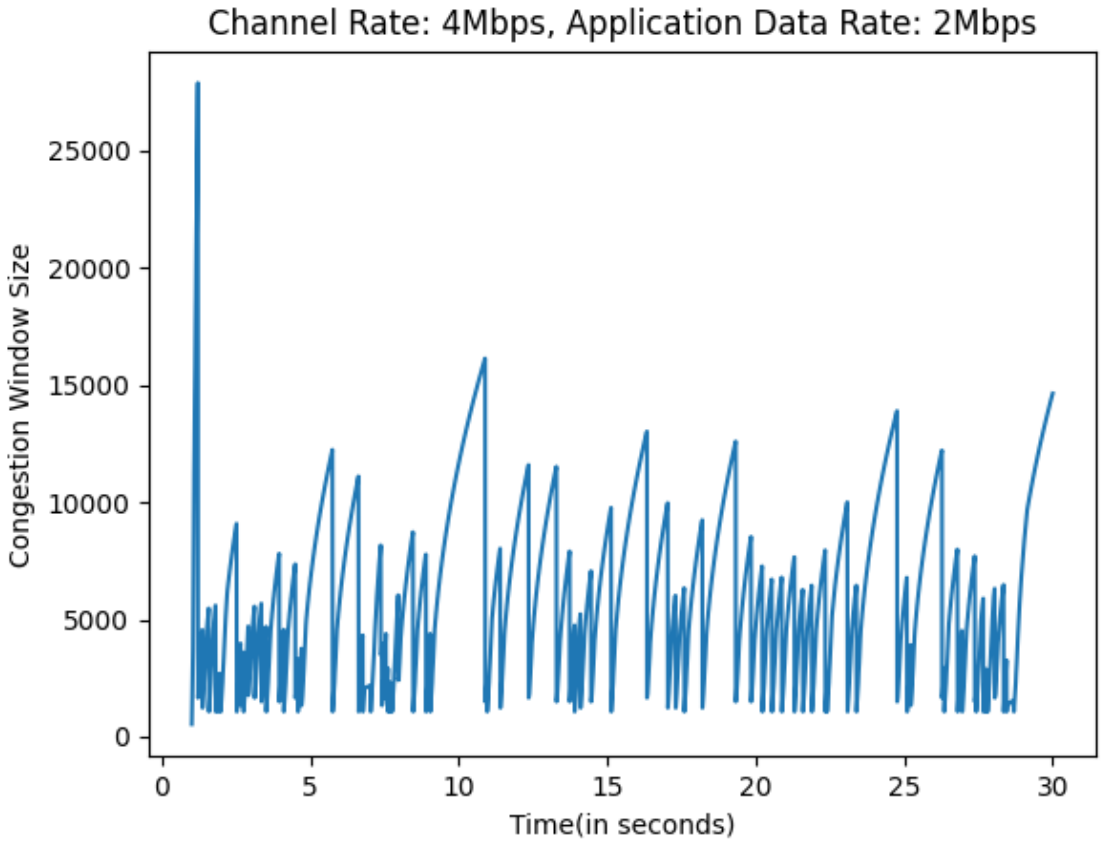
More Linear increase and decrease in window size.
The lower bound for new value when packets are dropped is higher (~2680) than TcpNewReno.
TcpVegas compares actual throughput with expected throughput (cwnd/BaseRTT) and regulates congestion window, so that diff = (actual - expected throughput) remains between threshold values.
Lowerst average cwnd size among all 4 plots

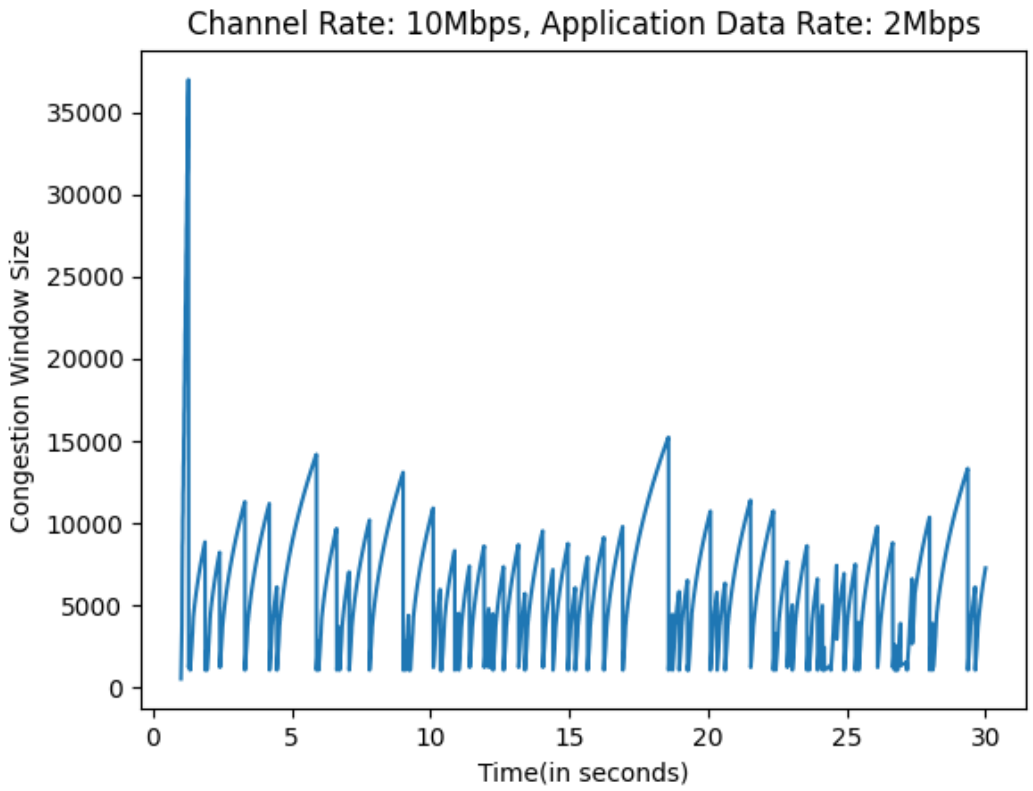
(d)TcpVeno

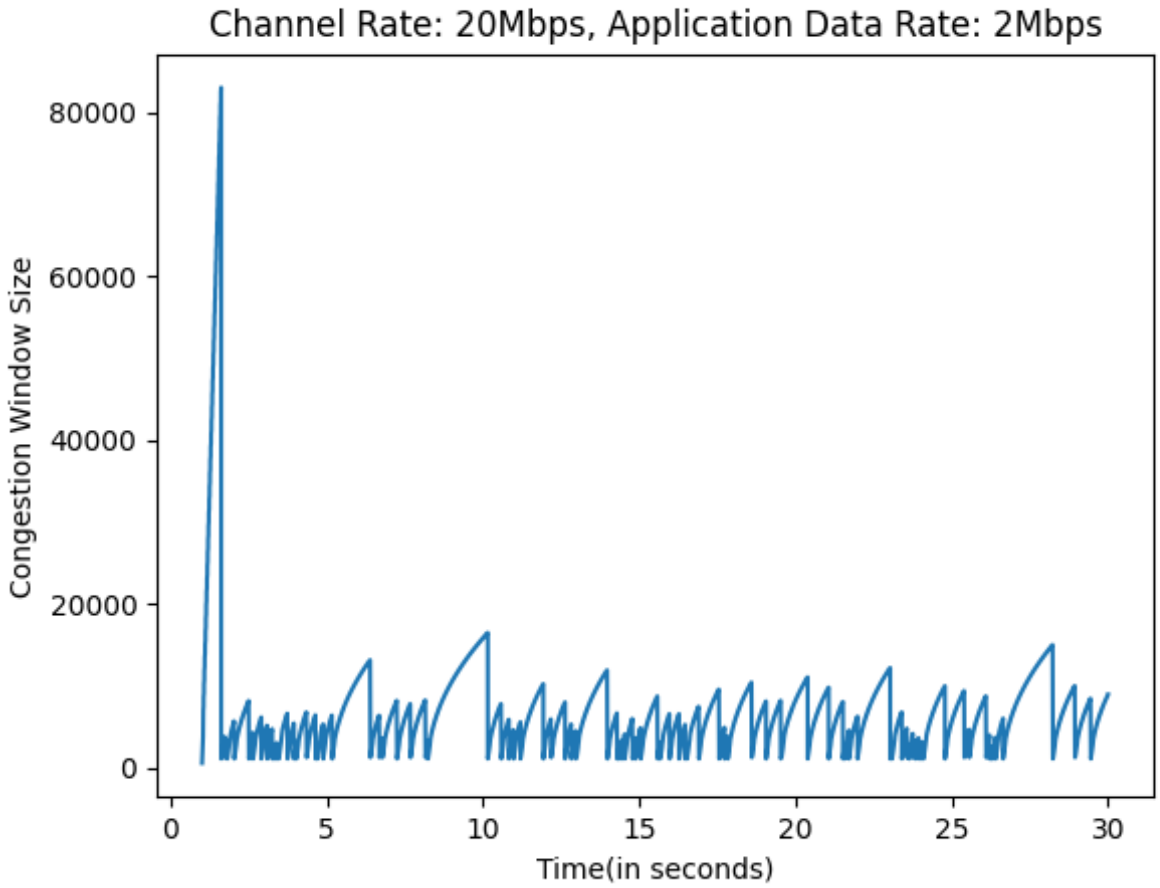
Congestion window drops to 1072 bytes for every packet drop.
Maximum congestion window size ~16000.
Highly similar to plot of TcpNewReno protocol.
In additive increase, cwnd is increase by (1/cwnd) for every slternate ACK received after the bandwidth is utilised.
If diff = cwnd/BaseRTT - cwnd/RTT > beta threshold, packet loss leads to multiplicative decrease by 1/2, else decrease is by 1/5.

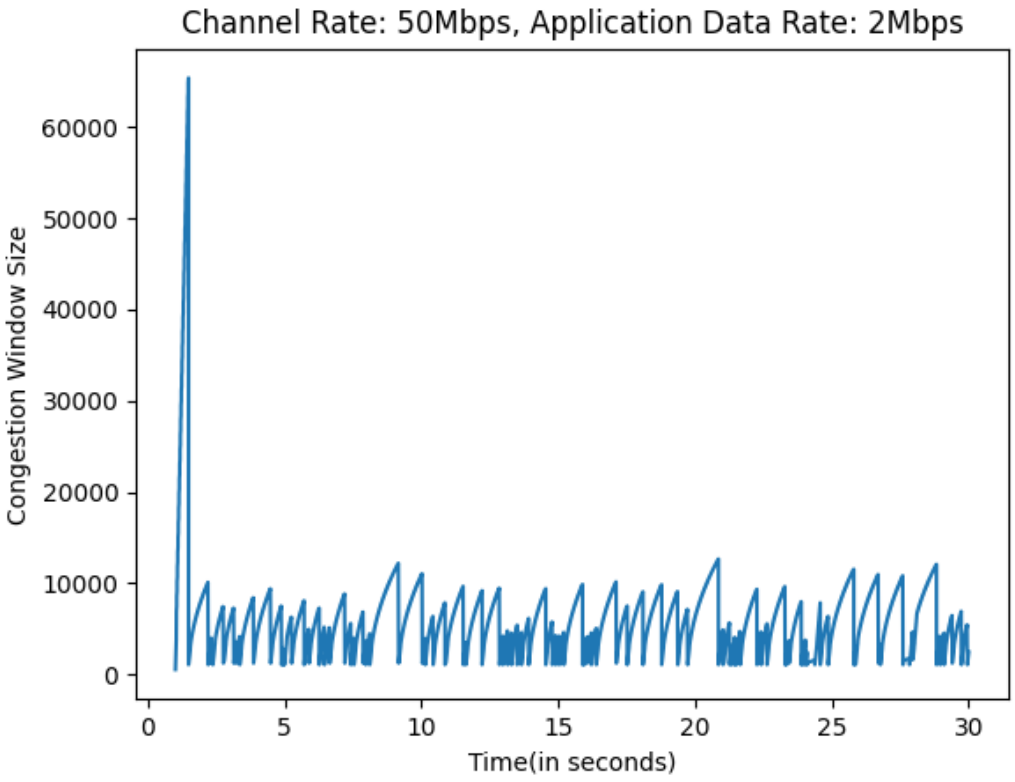
Q2) Plots for application rate: 2Mbps with varying channel rates 2, 4, 10, 20, 50 Mbps



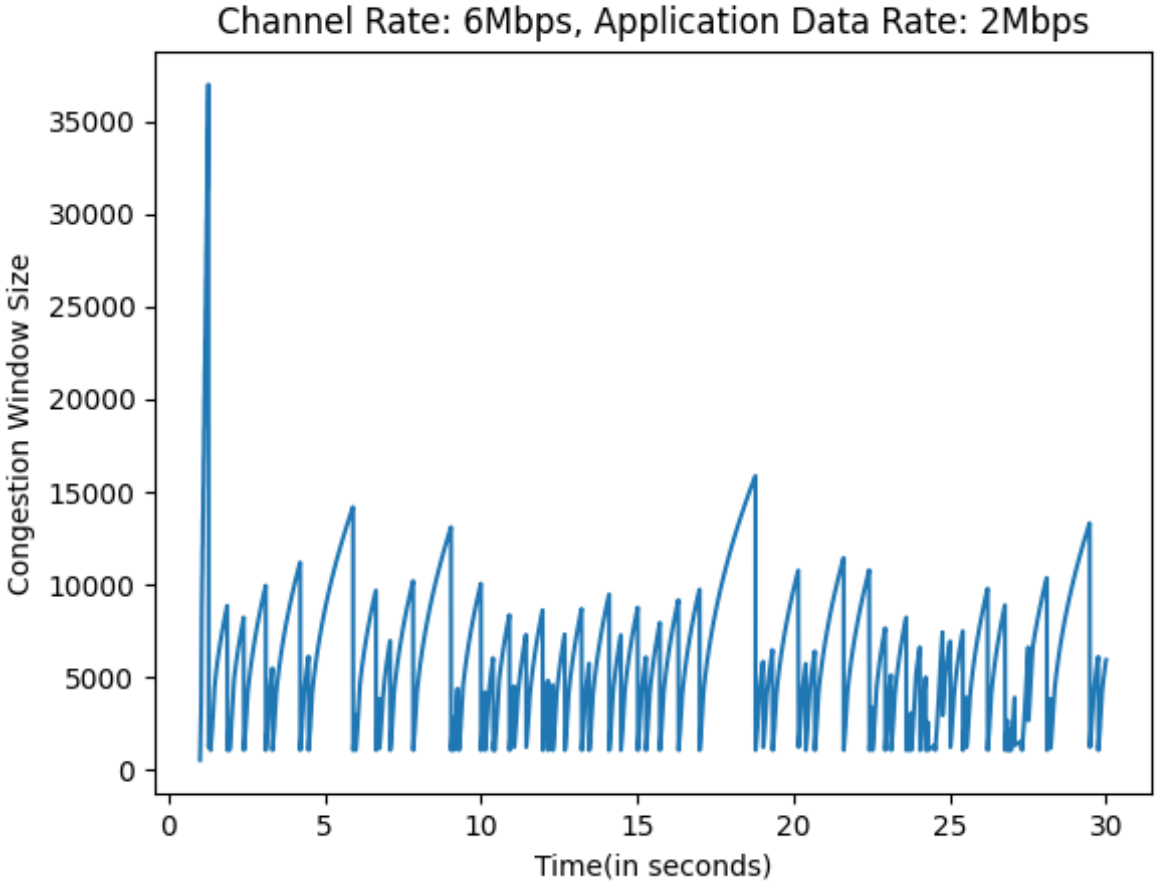


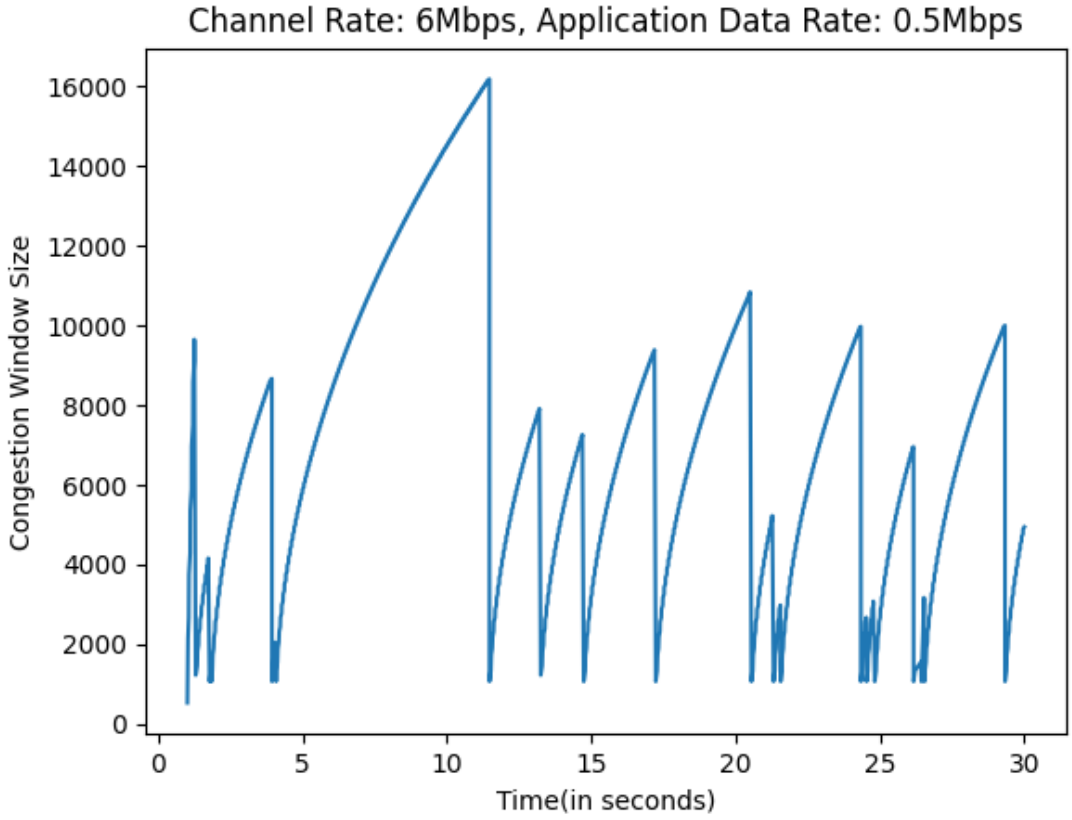

Channel Rate: 10Mbps, Application Data Rate: 2Mbps
Congestion Window Size
Time(in seconds)
0
5000
10000
15000
20000
25000
30000
35000
0
5
10
15
20
25
30

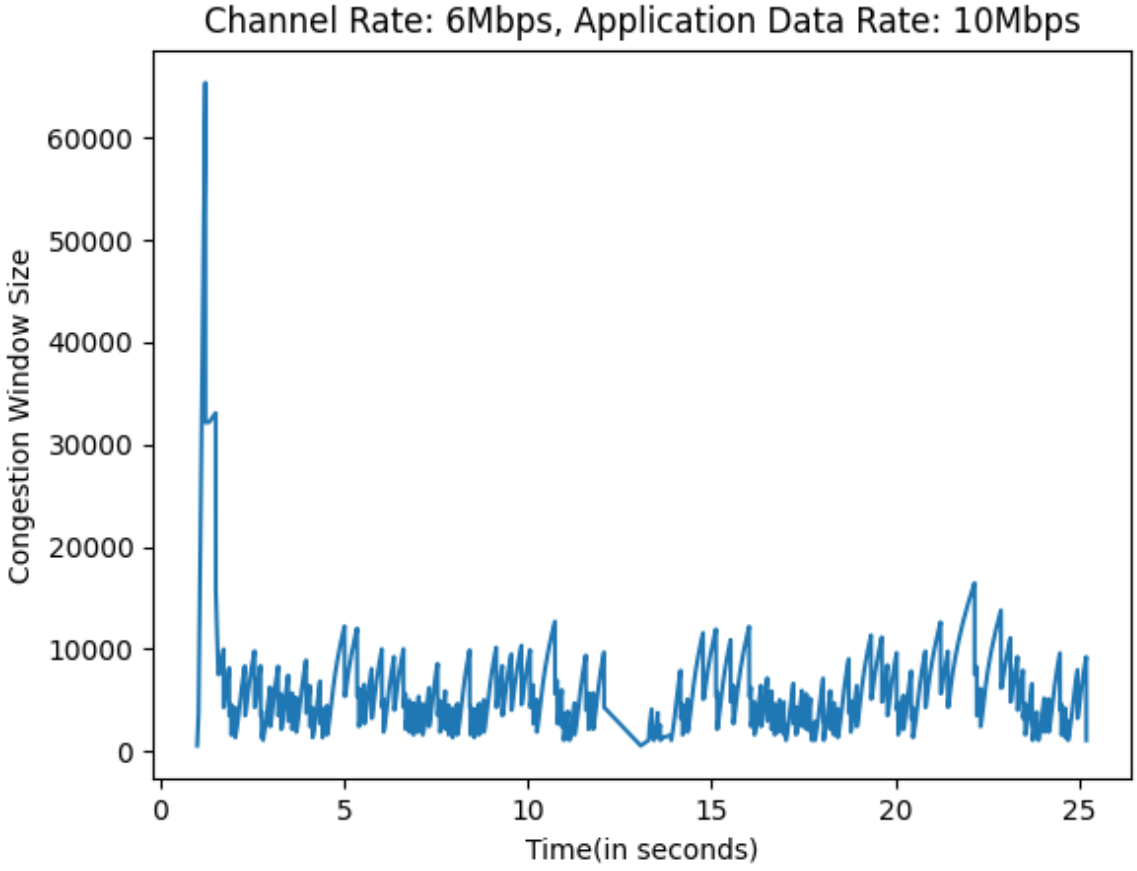

Channel Rate: 20Mbps, Application Data Rate: 2Mbps
Congestion Window Size
Time(in seconds)
0
20000
40000
60000
80000
0
5
10
15
20
25
30

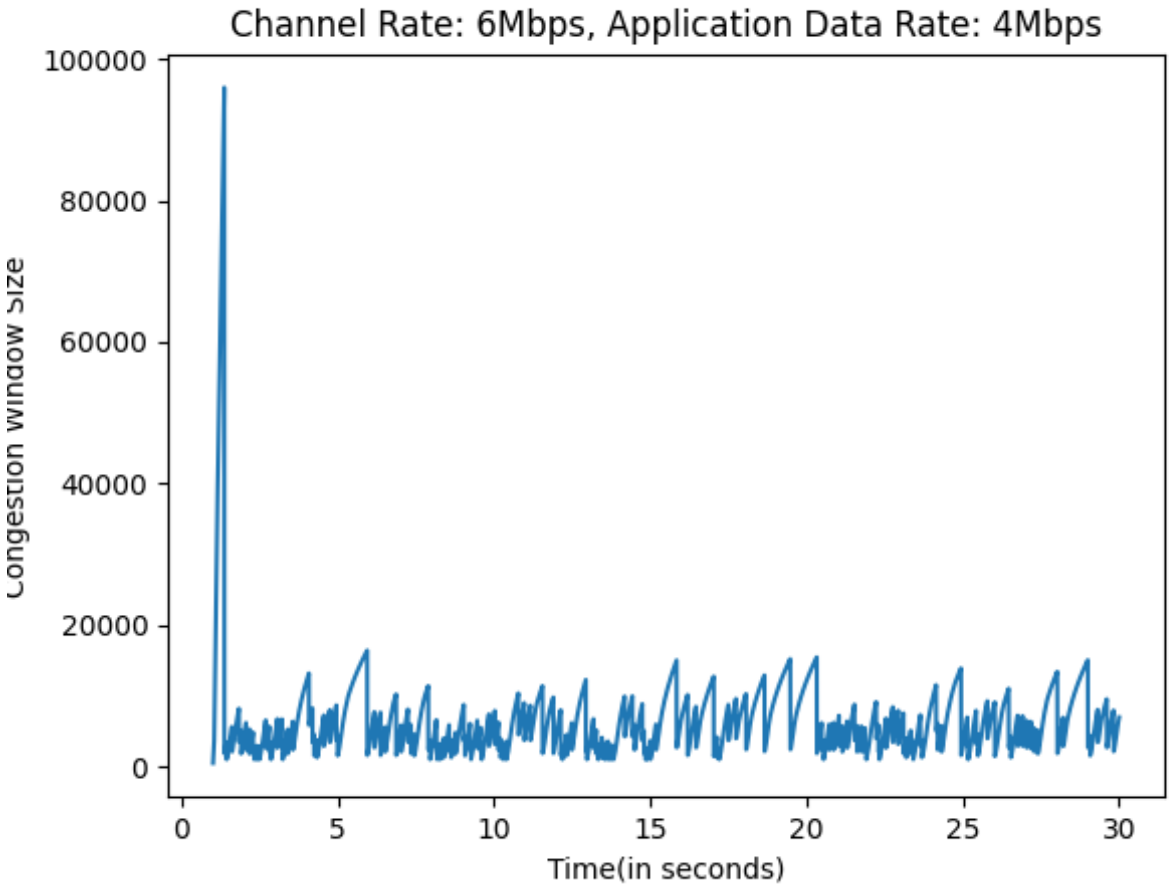


Plots for channel rate: 6Mbps with varying application rates 0.5, 1, 2, 4, 10 Mbps




Channel Rate: 6Mbps, Application Data Rate: 0.5Mbps
Congestion Window Size
Time(in seconds)
0
2000
4000
6000
8000
10000
12000
14000
16000
0
5
10
15
20
25
30


Channel Rate: 6Mbps, Application Data Rate: 10Mbps
Congestion Window Size
Time(in seconds)
0
10000
20000
30000
40000
50000
60000
0
5
10
15
20
25



Observations:

(A)

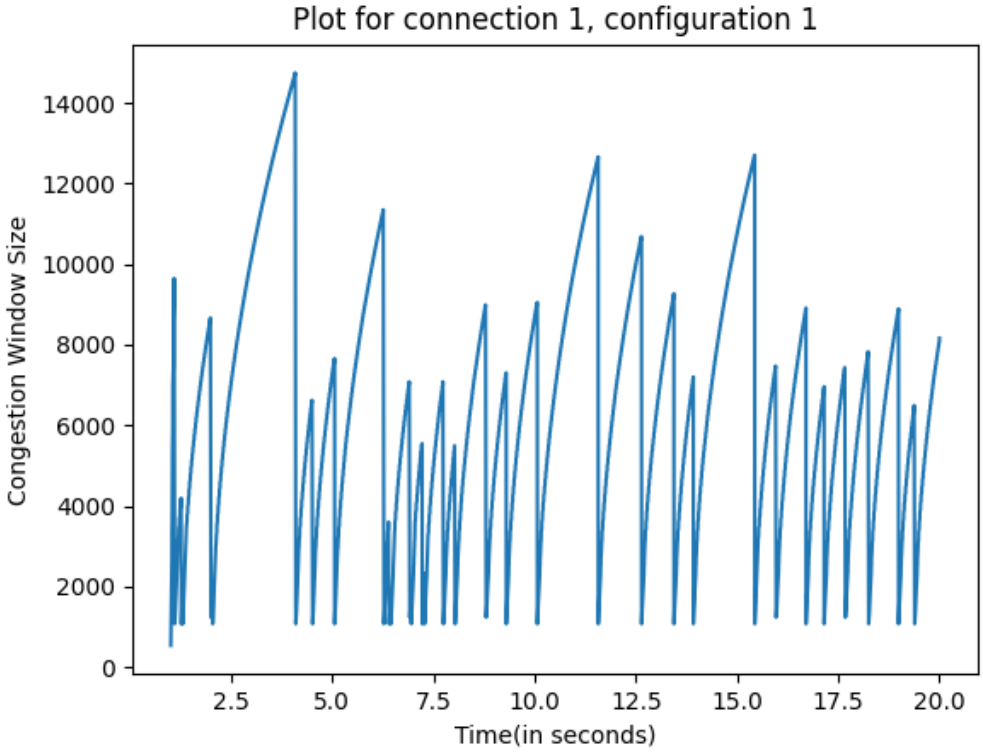
Increase in channel rate should intuitively leads to higher congestion window sizes. However, increased channel rates also lead to increase in dropped packets(Packets dropped by error model is proportional to channel rate). Thus, inspite of an increase in channel rate, the average congestion window size does not drastically increase

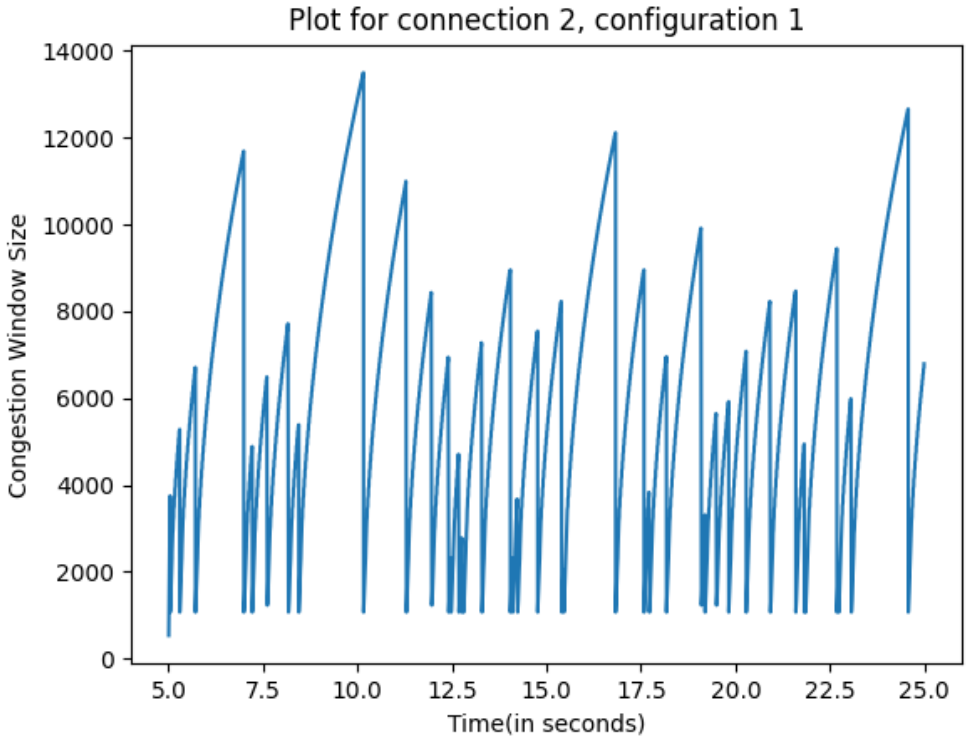
The peaks are higher and more in number in higher channel rates. Since more number of packets are sent at higher data rates in different channels, we observe that the data is transmitted better at higher channel data rates.

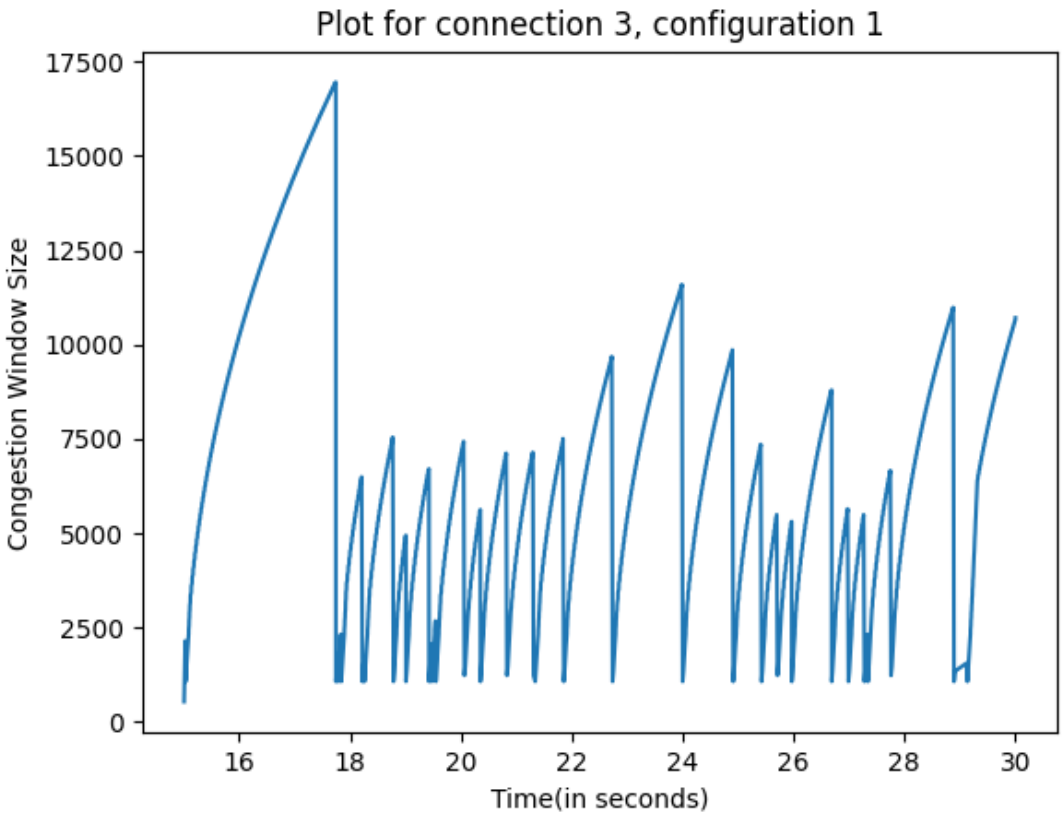
(B)

As the application data rate increases, packets sent increase. This leads to more number of dropped packets as error model is proportional to packet traffic. Similar to increase in channel rates, we also observe that the number of peaks are higher. This is because of better transmission of larger number of packets at higher application data rates. Larger flow of packets would lead to faster increase in cwnd size and therefore such peaks can be observed.

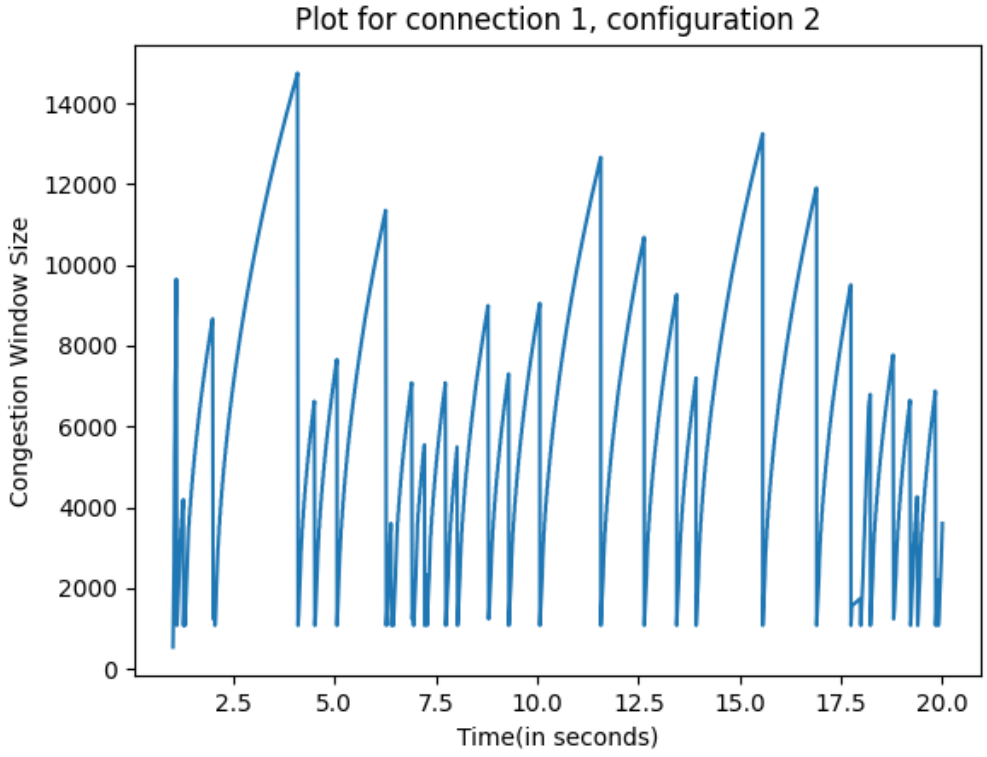
Q3)

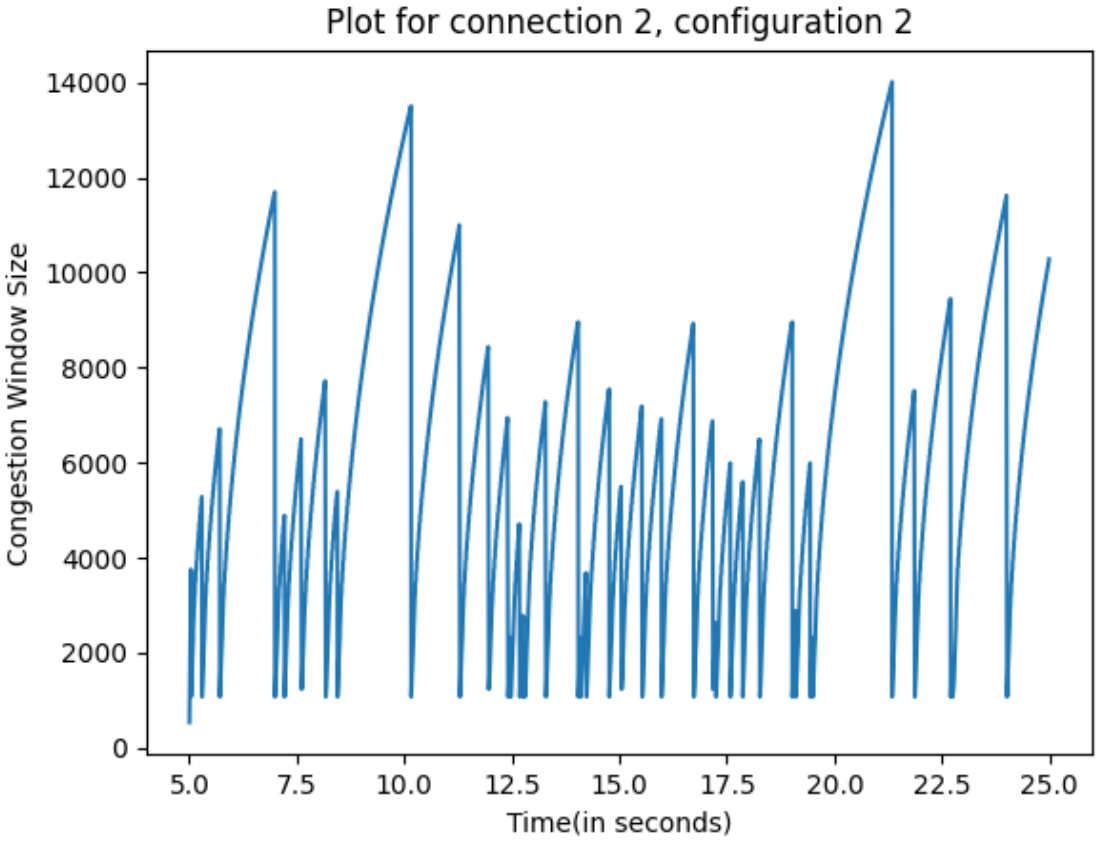

Plot for connection 1, configuration 1
Congestion Window Size
Time(in seconds)
0
2000
4000
6000
8000
10000
12000
14000
2.5
5.0
7.5
10.0
12.5
15.0
17.5
20.0

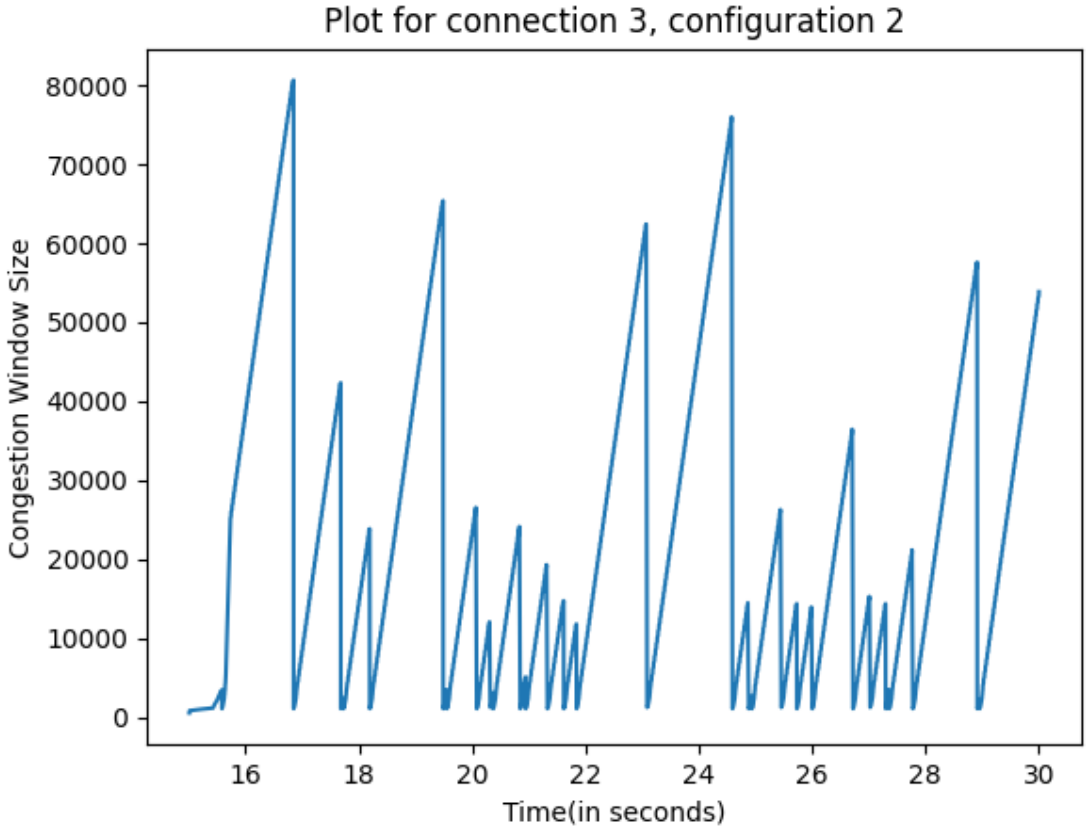

Plot for connection 2, configuration 1
Congestion Window Size
Time(in seconds)
0
2000
4000
6000
8000
10000
12000
14000
5.0
7.5
10.0
12.5
15.0
17.5
20.0
22.5
25.0



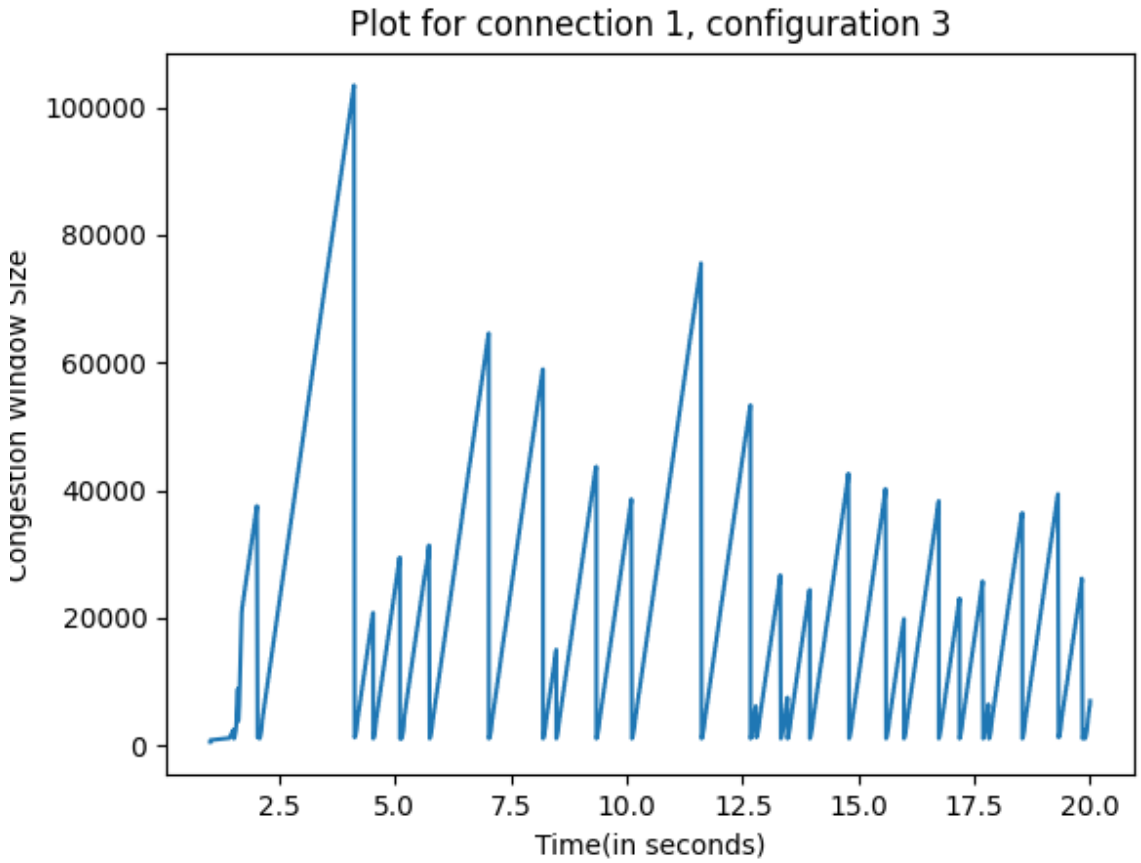
Plots for configuration 2:

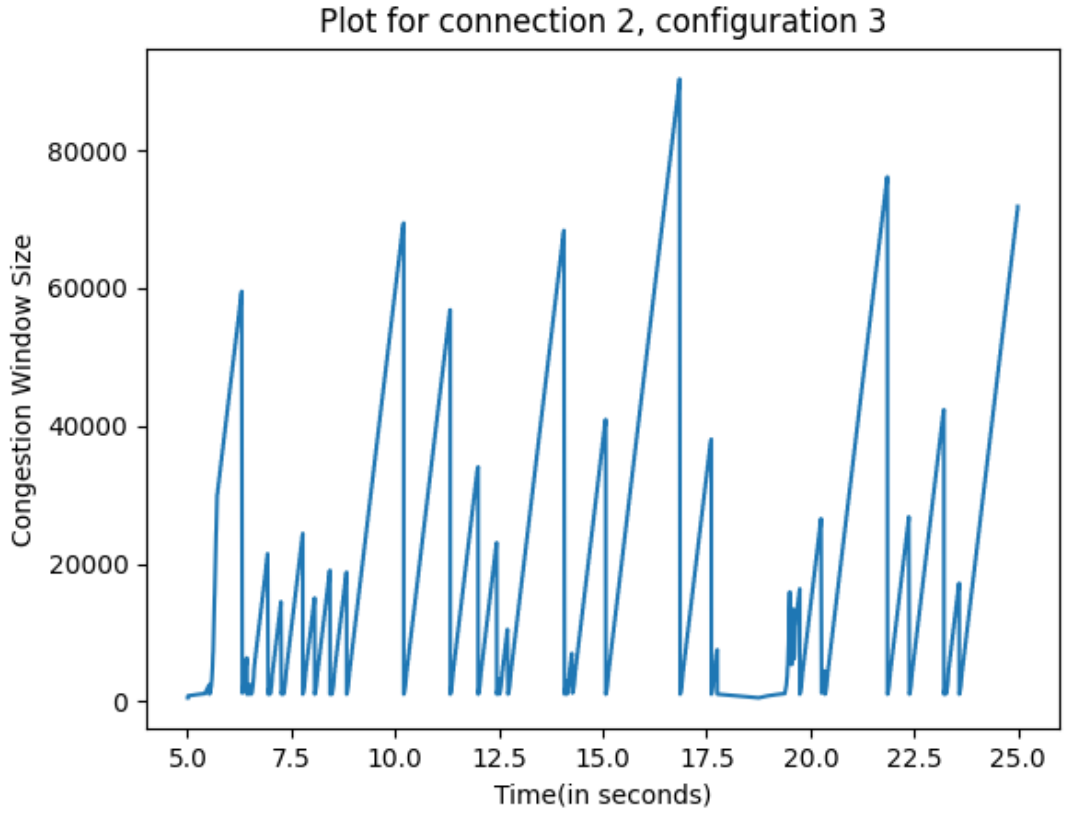


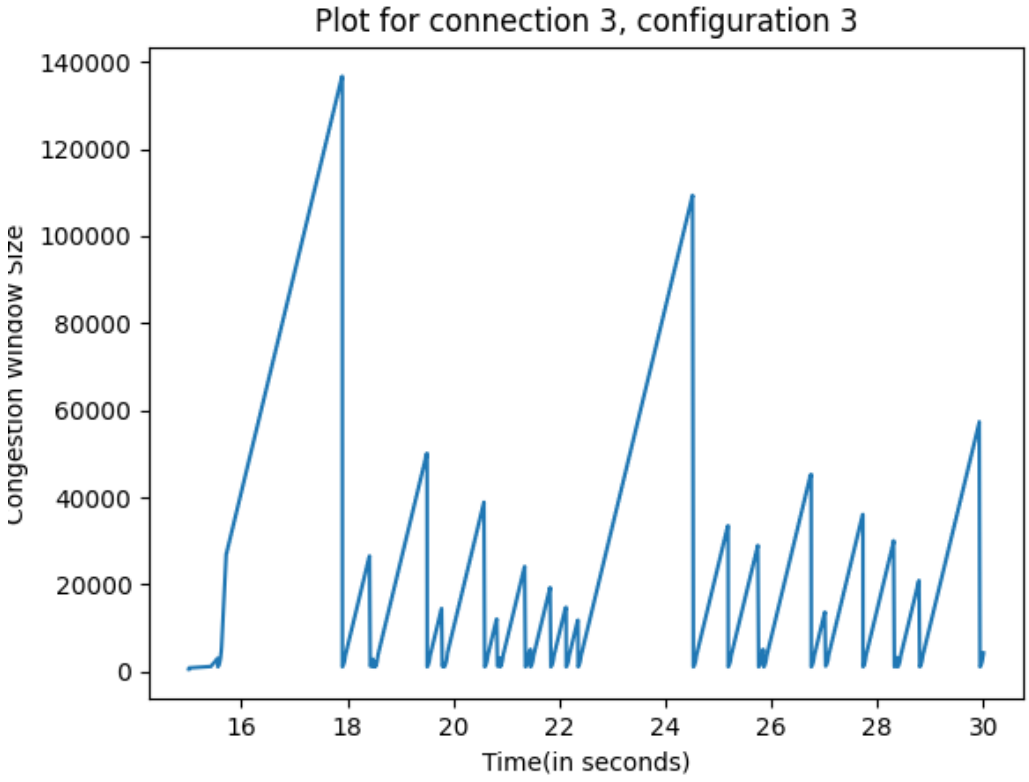

Plot for connection 2, configuration 2
Congestion Window Size
Time(in seconds)
0
2000
4000
6000
8000
10000
12000
14000
5.0
7.5
10.0
12.5
15.0
17.5
20.0
22.5
25.0


Plot for connection 3, configuration 2
Congestion Window Size
Time(in seconds)
0
10000
20000
30000
40000
50000
60000
70000
80000
16
18
20
22
24
26
28
30

Plots for configuration 3:







Dropped packet info: Config1: {N1 —- N3: 76, N2—-N3: 30}
Config2: {N1 —- N3: 80, N2—-N3: 30}
Config3: {N1 —- N3: 78, N2—-N3: }

Observation:

Configuration3 refers to when all senders follow TcpNewRenoCSE protocol.
Configuration2 refers to when only connection3 follows TcpNewRenoCSE.
Configuration refers to when all nodes follow TcpNewReno.

We see from the above plots that TcpNewRenoCSE gives maximum Window size ~80000, which is much largere than TcpNewReno(MaxWindowSize ~14000). This is because cwnd increment during slow start is proportional to segment size for TcpNewReno, whereas in TcpNewRenoCSE, increments is proportional to segment size^1.9. Hence, values are drastically higher with distinct high peaks.

During congestion Avoidance, the increment in TcpNewRenoCSE is linear whereas for TcpNewReno it is inversely proportional to cwnd. Hence, cwnd grows faster during congestion avoidance for TcpNewRenoCSE.
Number of packets dropped is also lesser for TcpNewRenoCSE

Files in Folder

Plots/: contains all plots attached in report

Scripts/: contains all required scripts

Note:

Add TcpNewRenoCSE.h, TcpNewRenoCSE.cc to ns-3.29/src/internet/models and update required wscript file. Place all other scripts in scratch/

run first_1.cc with args '—TcpL4Protocol=<protocol>'

Run third.cc with args '—configuration=<1, 2, or 3>'