COL334 Assignment 3

Arpit Saxena, 2018MT10742

Contents

How to Run

To run, place the First.cc, Second.cc, Third.cc in ns3's scratch/ folder and place the `TCPNewRenoCSE{.h,.cc}` files in the proper folder and fix the `wscript` to include them. Certain directories are also expected. Make the following directory structure in the root of where output will be:

```
tracesFirst/
tracesSecond/
    a/
    b/
tracesThird
```

Submitted Files Description

- Q1/First.cc : Consists of code for Part 1. Most of the `MyApp` code is taken from ns3's `examples/tutorial/sixth.cc`.
- Q2/Second.cc : Consists of code for Part 2. Most of the `MyApp` code is taken from ns3's `examples/tutorial/sixth.cc`.
- Q3/Third.cc : Consists of code for Part 3. Most of the `MyApp` code is taken from ns3's `examples/tutorial/sixth.cc`.
  - Q3/Congestion/TCPNewRenoCSE.h : Declares class `TCPNewRenoCSE`, inheriting it from `TCPNewReno`
  - Q3/Congestion/TCPNewRenoCSE.cc : Implements `TCPNewRenoCSE`, implementing the overridden methods `SlowStart` and `CongestionAvoidance`
- plotTraces.py : Contains code to plot traces. Run `python plotTraces.py --help` for details. It basically takes a folder, and for each fileName.cwnd (containing time vs congestion window size), it expects fileName.drops containing times of packet drops and plots the congestion window size vs time and saves the plot as fileName.png.

Part 1

Figure 1, Figure 2, Figure 3 and Figure 4 show the congestion window size vs time for different congestion control protocols. They also mention the number of dropped packets for each protocol, which is also summarised in Table 1

Table 1: Number of packet drops in different congestion control protocols

Congestion control method	Packet Drops	Total number of packets sent
NewReno	37	1209
HighSpeed	37	1209

Congestion control method	Packet Drops	Total number of packets sent
Veno	34	1209
Vegas	50	1209

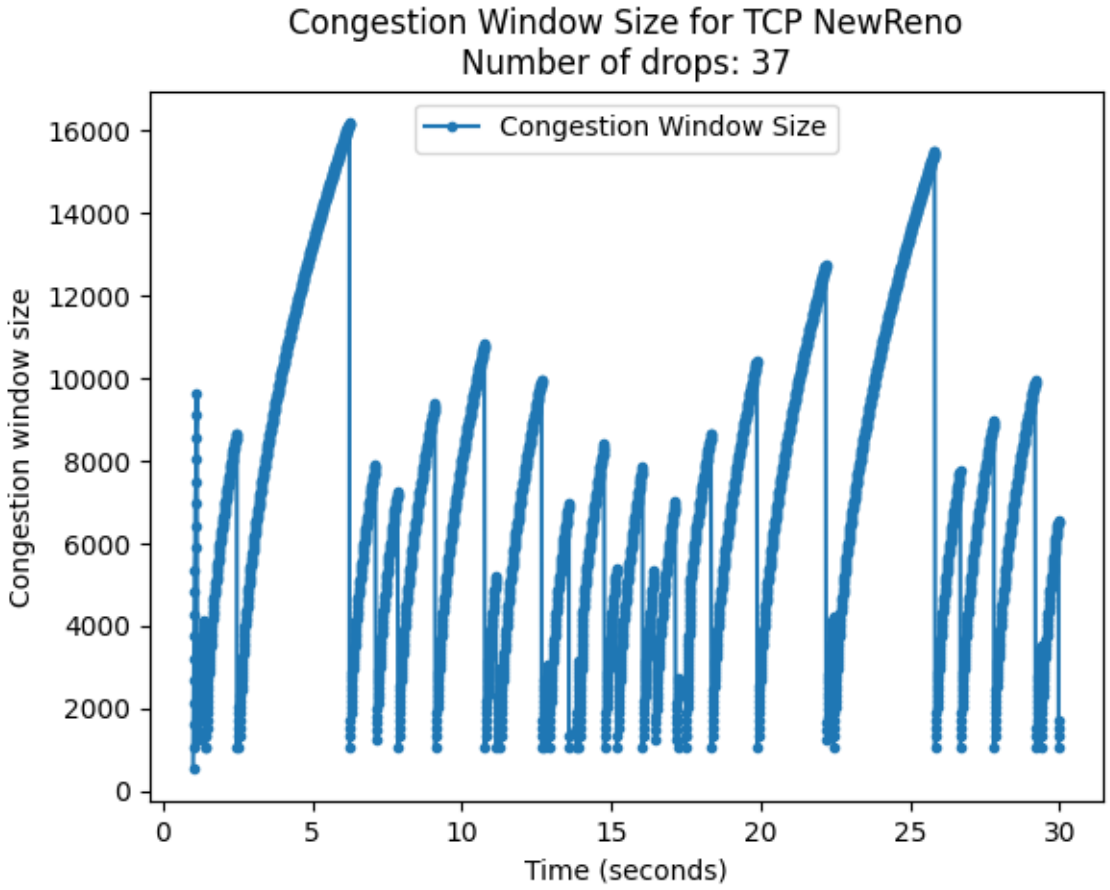

Figure 1: Congestion Window Size vs Time for TCP New Reno

Observations regarding the different protocols are as follows:

1. NewReno:
   - It is based on the Reno algorithm and introduces partial ACKs in it
   - Like Reno, it enters into fast retransmit mode when it receives multiple duplicate ACKs, but it doesn't exit until all the ACKs remaining at the time it entered are received
   - In Figure 1, we can observe a very rapid increase in the beginning when cwnd reaches about 10,000 in a very short interval. That marks the slow start phase
   - After that, it entered into congestion avoidance phase where `cwnd` was increased slowly until a loss event happened.
2. HighSpeed:
   - It is designed for high-capacity channels, or in general, for TCP connections with large congestion windows.
   - With respect to standard TCP, it makes `cwnd` grow faster but only when the window is a bove some threshold, so it remains friendly with standard TCP in high congestion cases.
   - In Figure 2, we can observe `cwnd` went to around 50,000 bytes in the slow start phase.
   - In congestion avoidance phases, near the higher peaks, we can observe the convexity of the graph changing which shows how HighSpeed increases cwnd rapidly above a certain threshold.
3. Vegas:
   - It is a pure delay-based congestion control algorithm that tries to prevent packet drops by maintaining a small backlog of packets.

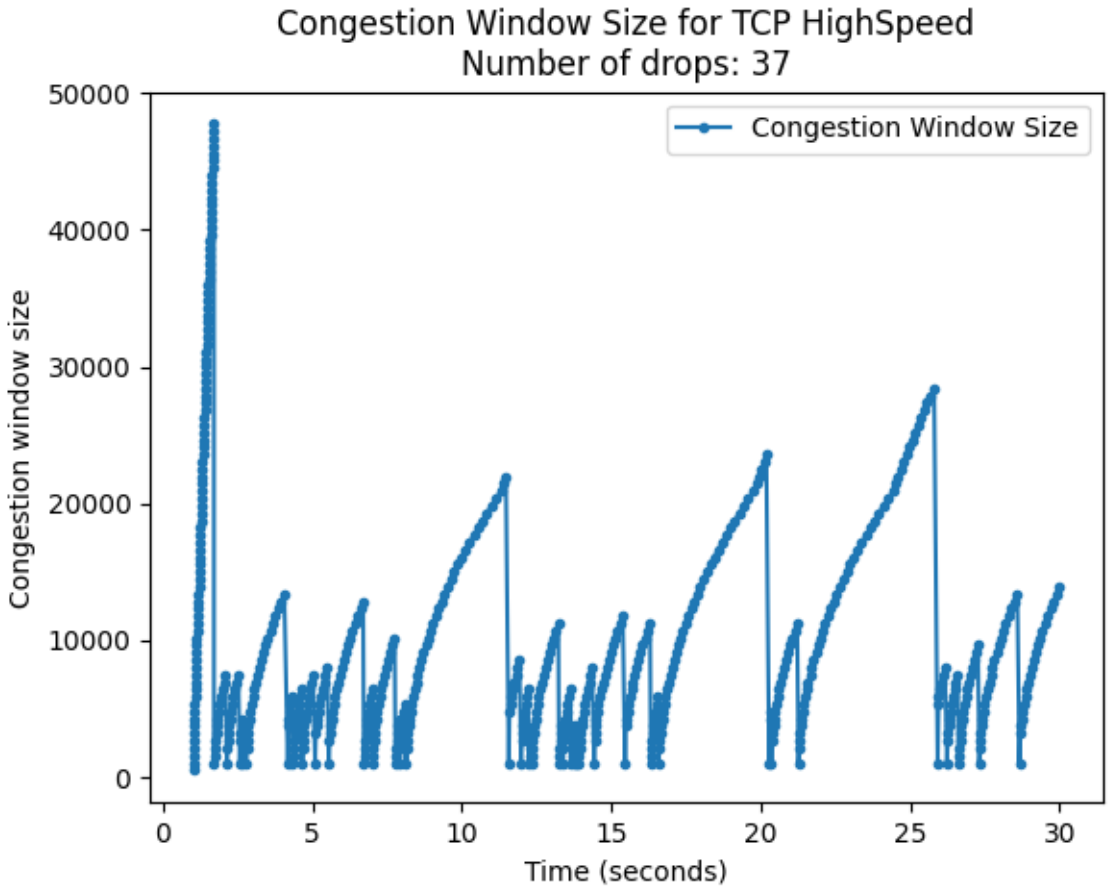

Figure 2: Congestion Window Size vs Time for TCP High Speed

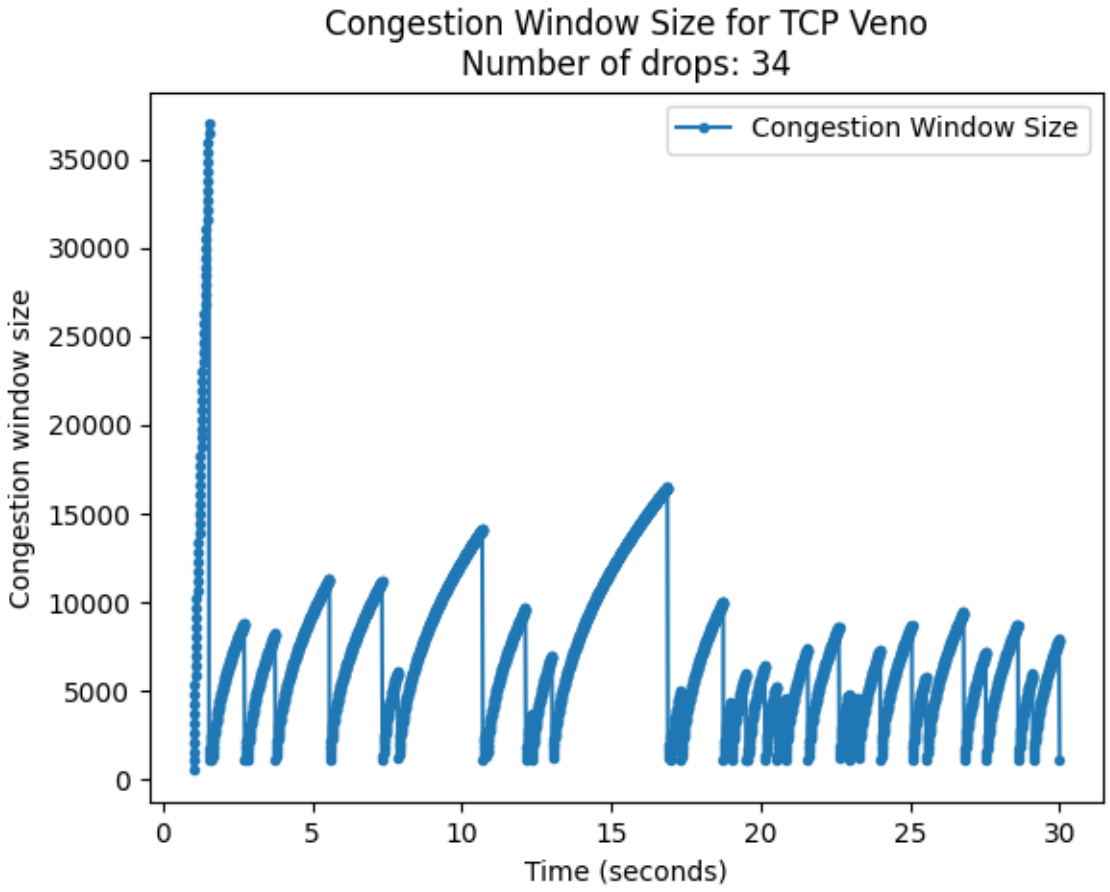

Figure 3: Congestion Window Size vs Time for TCP Veno

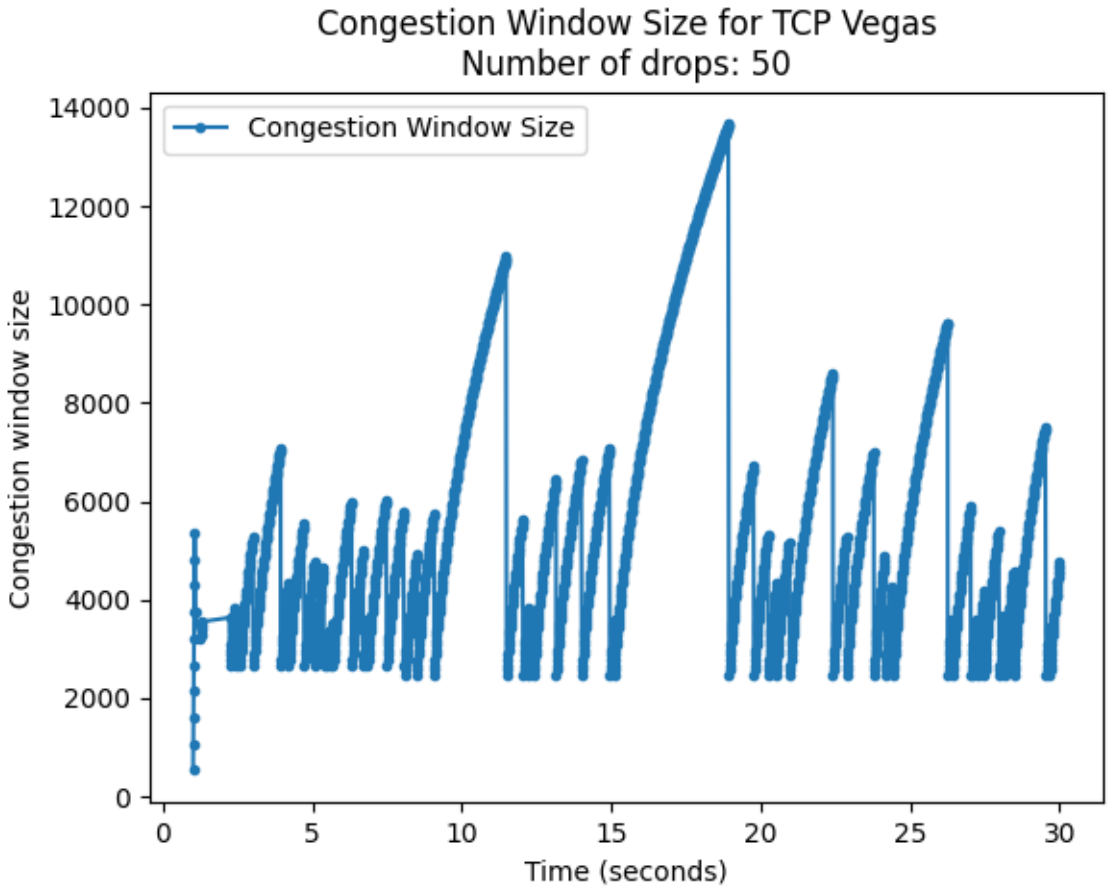

Figure 4: Congestion Window Size vs Time for TCP Vegas

- It continuously samples the RTT and computes the actual throughput a connection achieves and by comparing it with the expected throughput, it holds back some packets in the bottleneck.
- In Figure 4, we can observe `cwnd` went to around 6,000 bytes in the initial phase.
- An interesting thing to not here is that when the congestion window size is decreased, it isn't set all the way back to 1 like it is done in other methods.

4. Veno:
   - It is also based on Reno, and designed to distinguish between random packet loss in wireless networks and losses due to congestions.
   - It employs Vegas's method in estimating backlog at the bottleneck queue to distinguish between congestive and non-congestive states.
   - In Figure 3, we can observe `cwnd` increased to 35,000 in the slow start phase which is the second highest among all of these methods.
   - The graph for the congestion avoidance phase looks very similar to that of TCP NewReno.

Part 2

Part a

Figure 5, Figure 6, Figure 7, Figure 8 and Figure 9 show the congestion window size vs time for varying channel data rates. They also mention the number of dropped packets for each protocol, which is also summarised in Table 2

Table 2: Number of packet drops with different channel data rates

Channel data rate	Packet Drops
2 Mbps	65
4 Mbps	75
10 Mbps	74
20 Mbps	88
50 Mbps	95

Observations:

- Number of packet drops increases as the channel data rate increases. As channel data rate increases, we are able to send (and receive) more packets in a given time span so increase to `cwnd` is quicker. This means we reach the congestion more quickly, and so the drops are more.
- Time between decrease of `cwnd` decreases when channel rate is increased. This can also be explained by being able to send more packets in a given time.
- We can also observe some fast retransmit cases where when channel data rate was 2 Mbps, where `cwnd` was not reduced all the way back to 1.

Part b

Figure 10, Figure 11, Figure 12, Figure 13 and Figure 14 show the congestion window size vs time for varying application data rates. They also mention the number of dropped packets for each case, which is also summarised in Table 4

Table 3: Number of packet drops with different application data rates

Application data rate	Packet Drops
0.5 Mbps	16
1 Mbps	35
2 Mbps	82
4 Mbps	143
10 Mbps	178

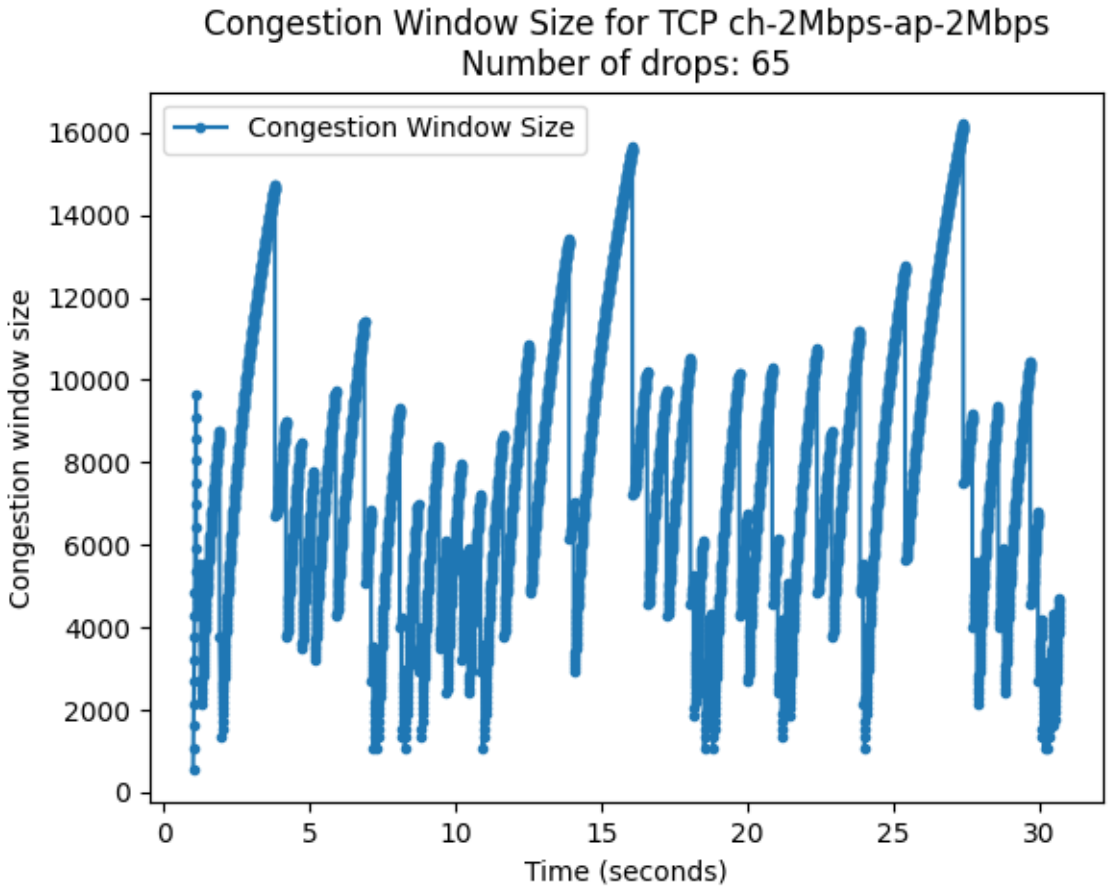

Figure 5: Congestion Window Size vs Time for Channel Rate of 2 Mbps and Application rate of 2Mbps

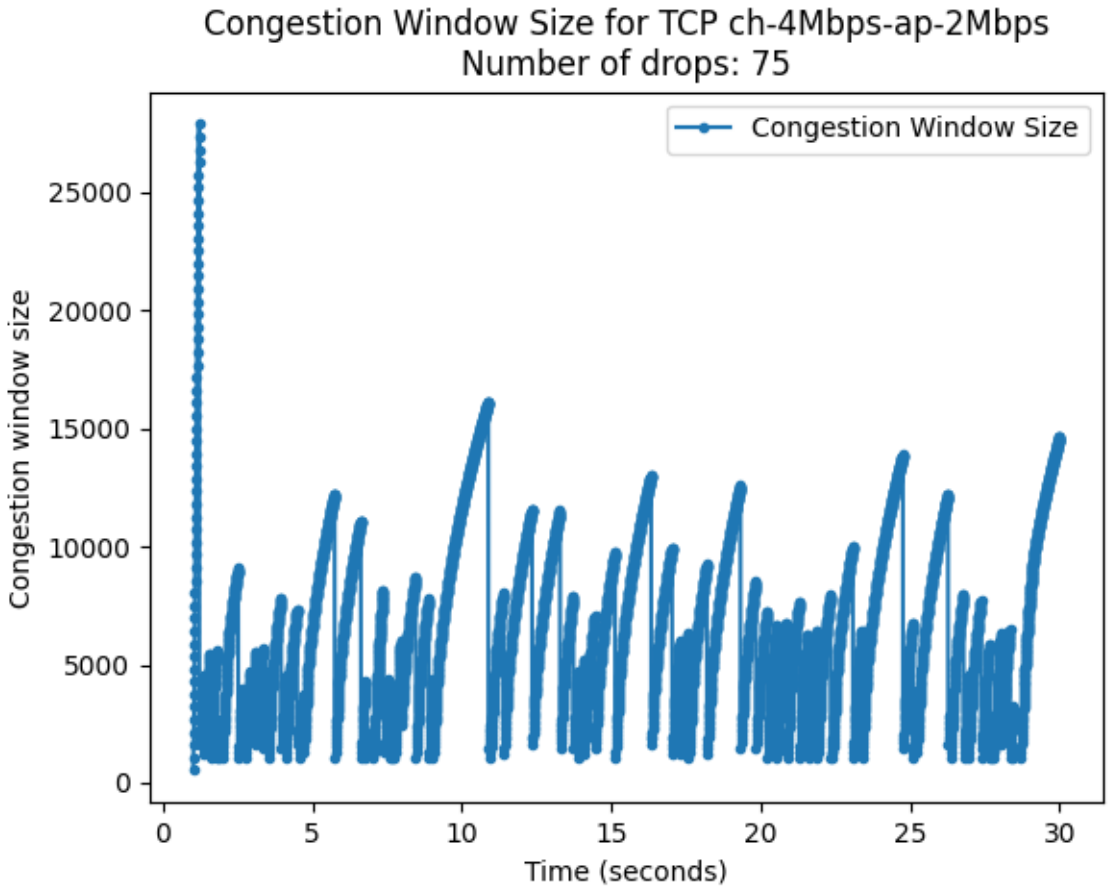

Figure 6: Congestion Window Size vs Time for Channel Rate of 4 Mbps and Application rate of 2Mbps

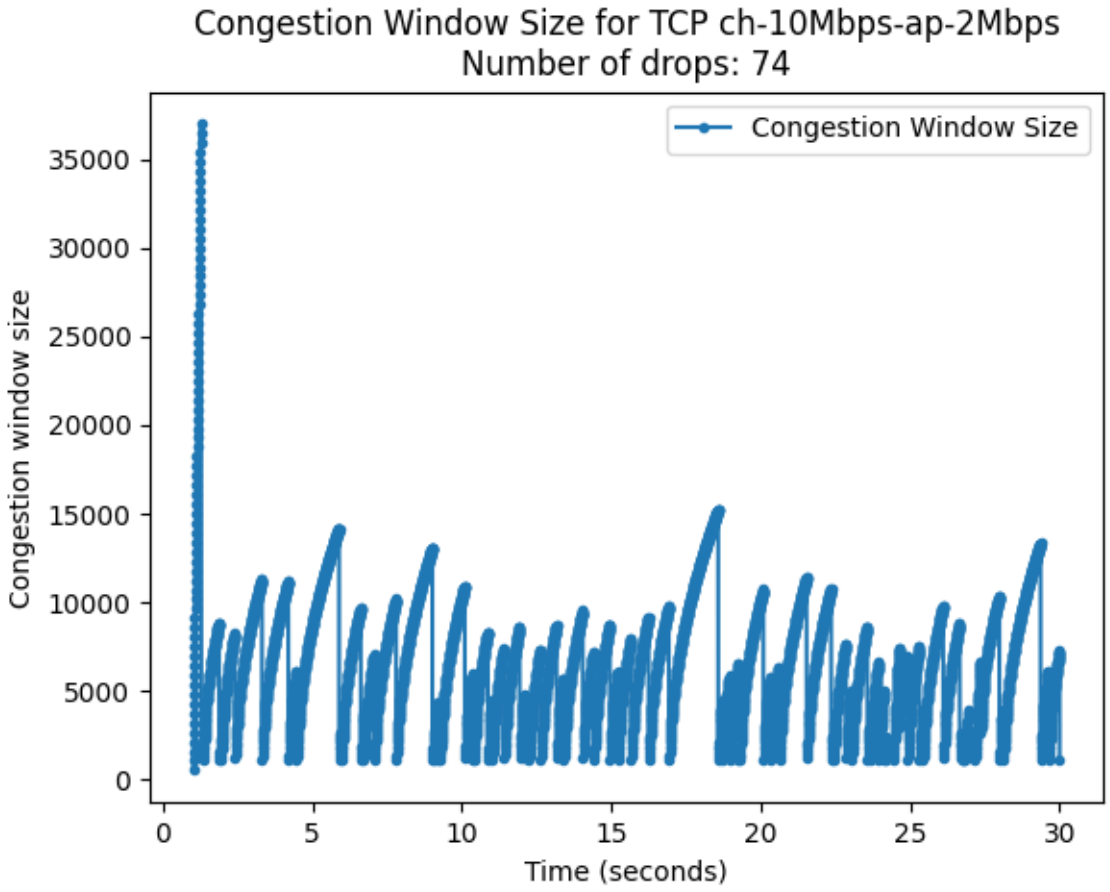

Figure 7: Congestion Window Size vs Time for Channel Rate of 10 Mbps and Application rate of 2Mbps

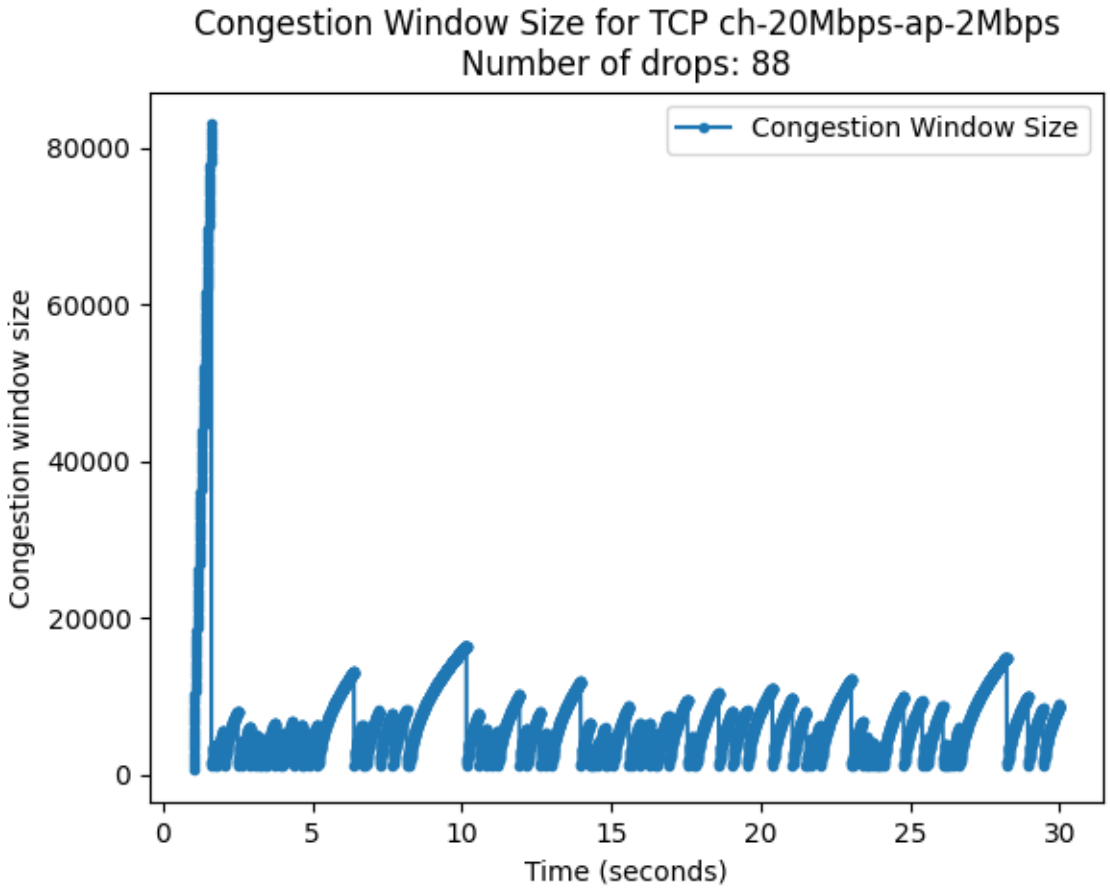

Figure 8: Congestion Window Size vs Time for Channel Rate of 20 Mbps and Application rate of 2Mbps

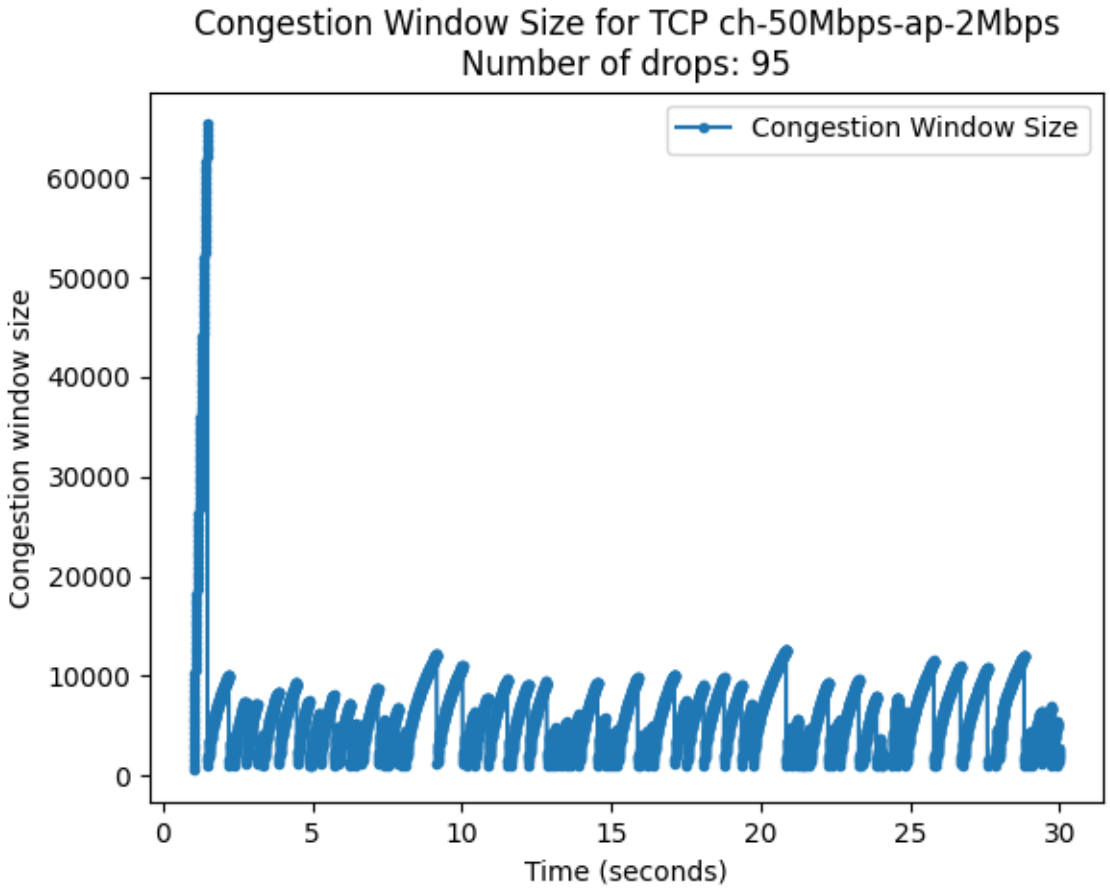

Figure 9: Congestion Window Size vs Time for Channel Rate of 50 Mbps and Application rate of 2Mbps

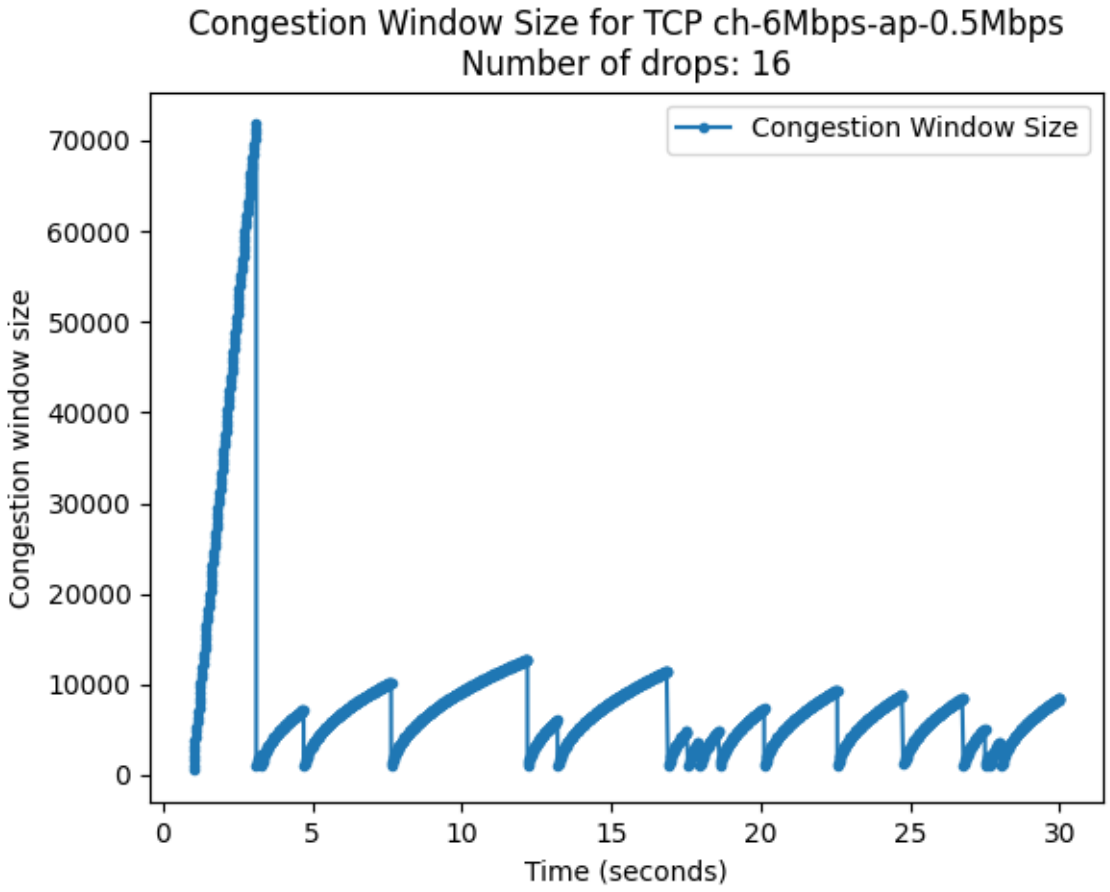

Figure 10: Congestion Window Size vs Time for Channel Rate of 6 Mbps and Application rate of 0.5 Mbps

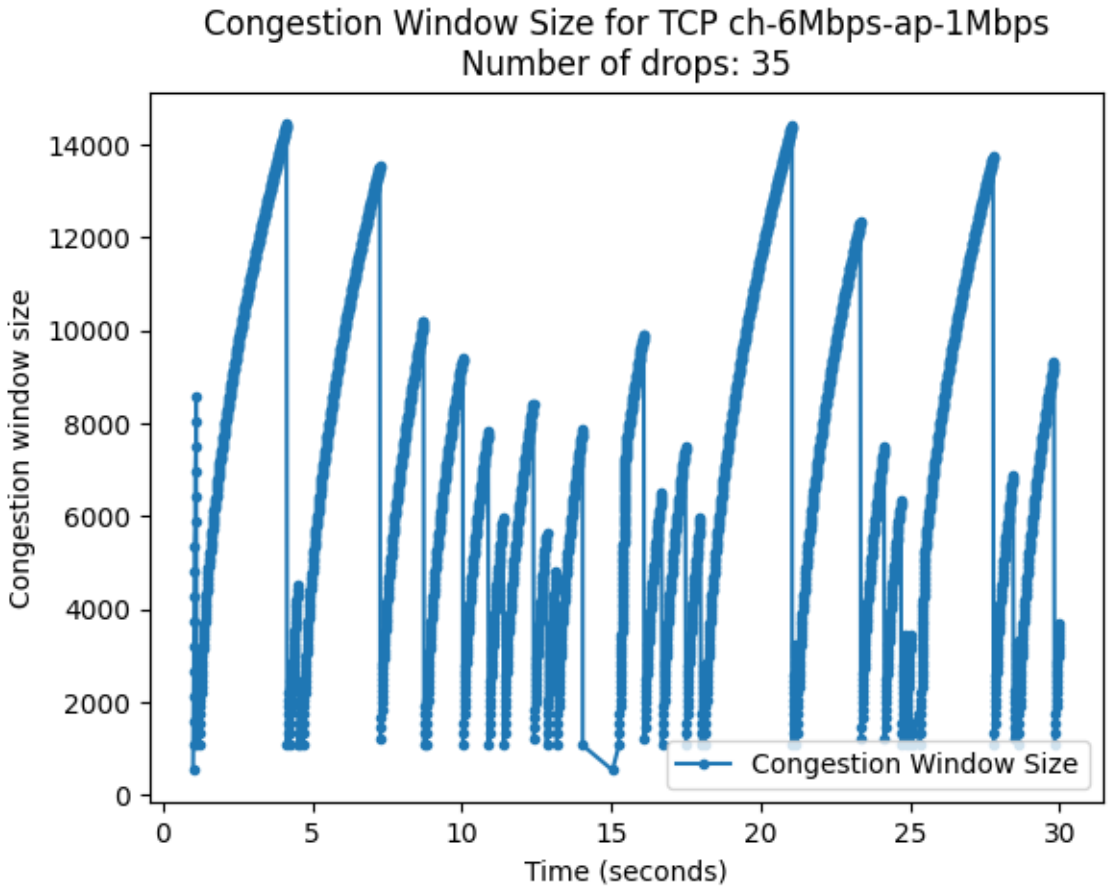

Figure 11: Congestion Window Size vs Time for Channel Rate of 6 Mbps and Application rate of 1 Mbps

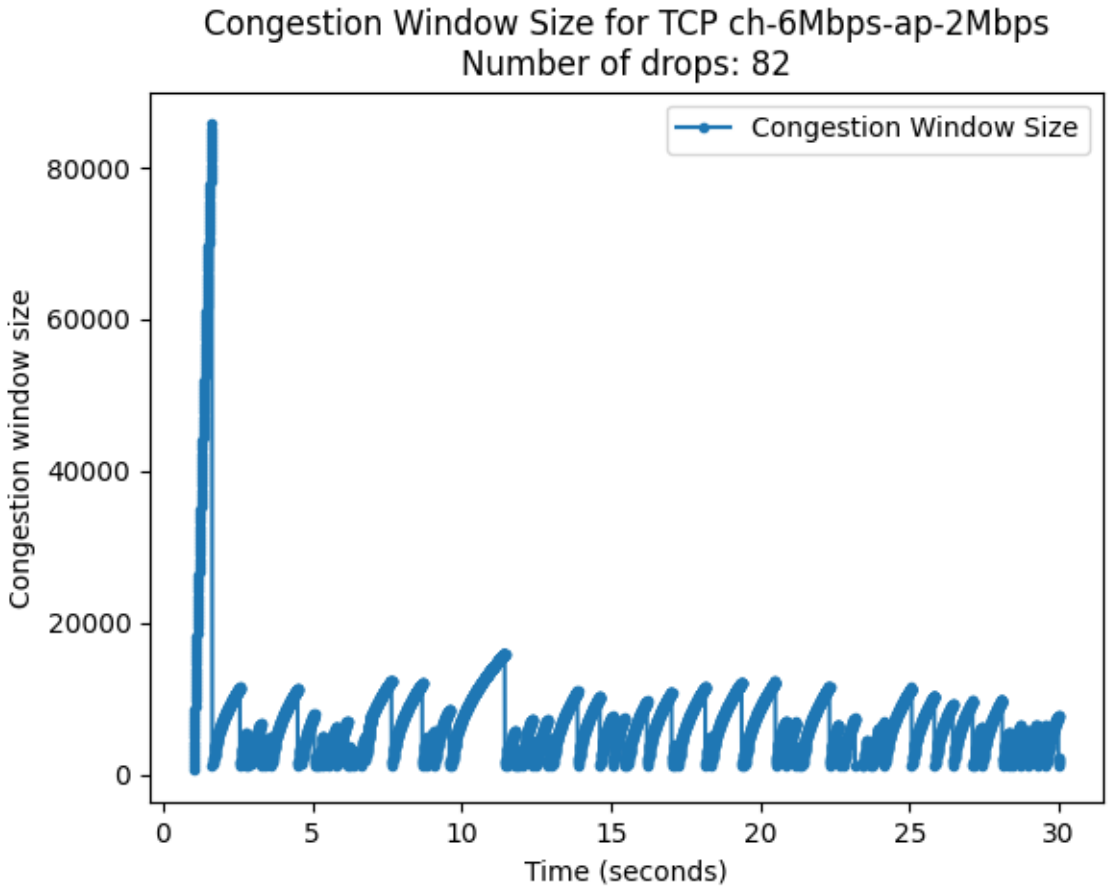

Figure 12: Congestion Window Size vs Time for Channel Rate of 6 Mbps and Application rate of 2 Mbps

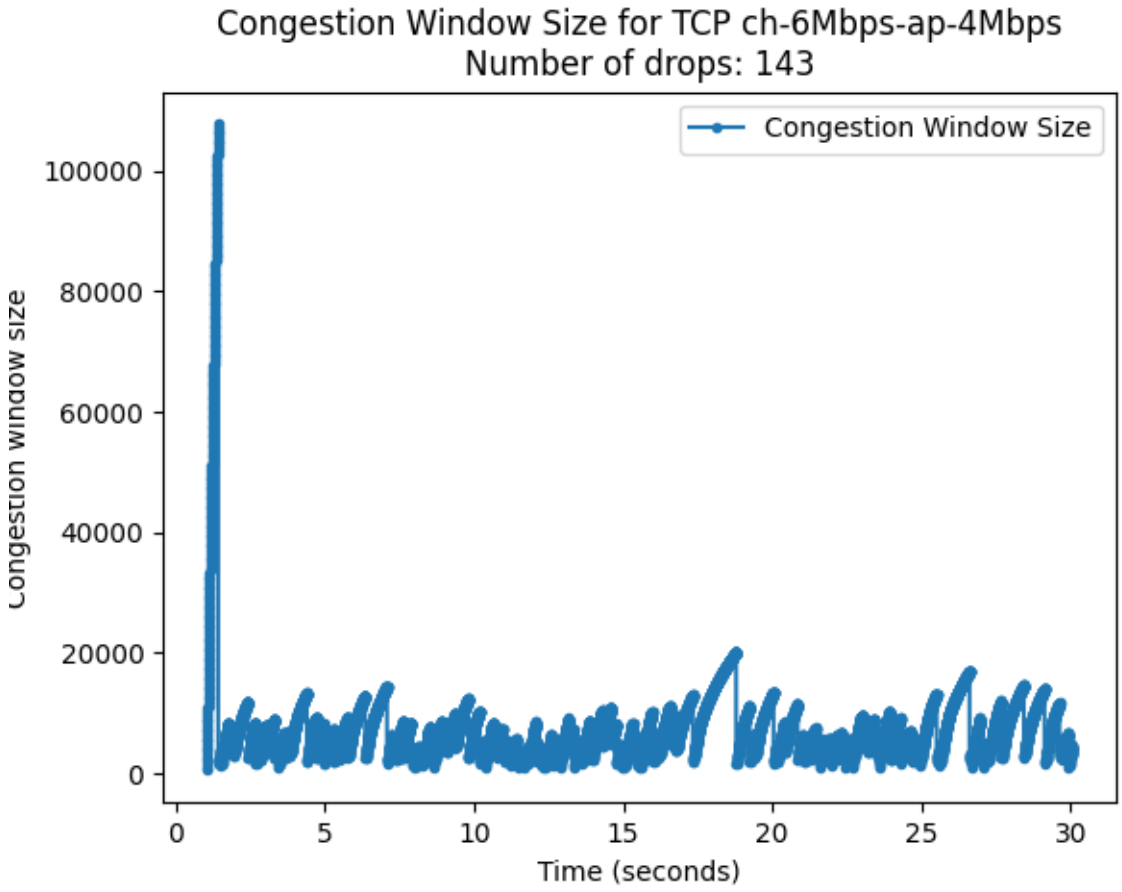

Figure 13: Congestion Window Size vs Time for Channel Rate of 6 Mbps and Application rate of 4 Mbps

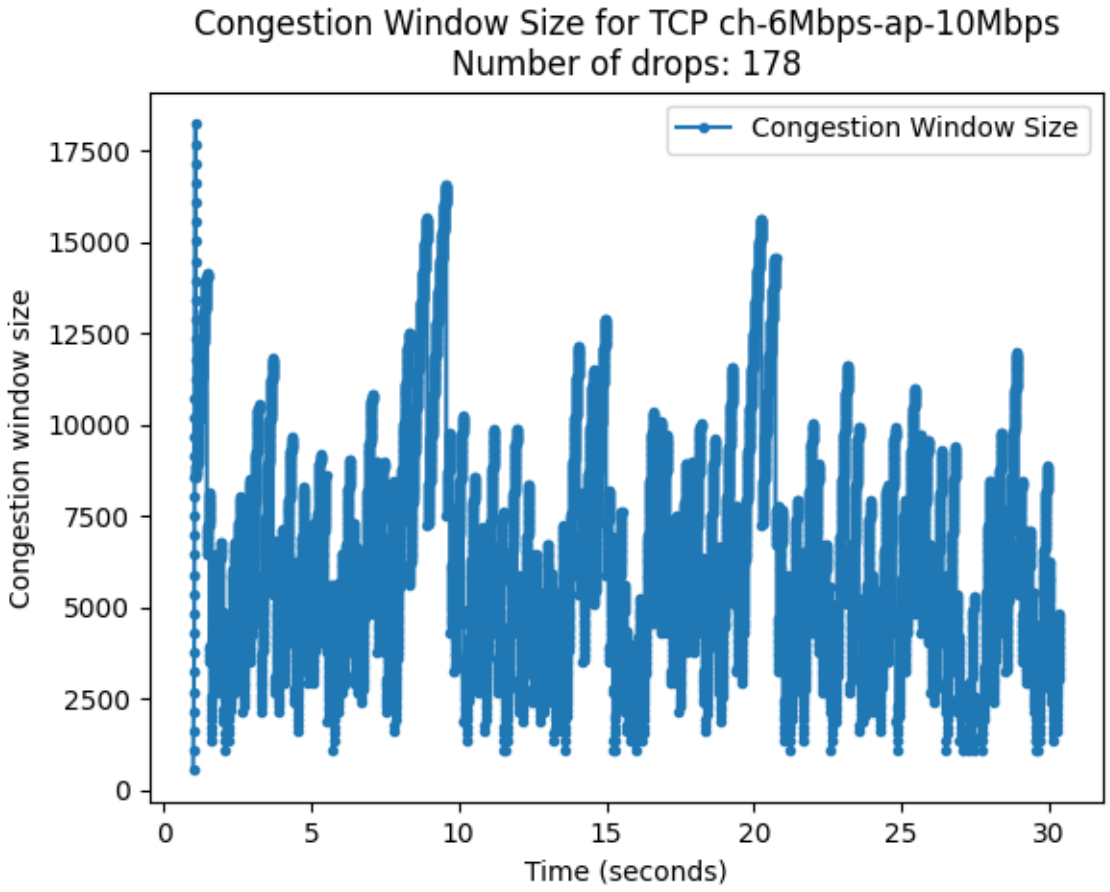

Figure 14: Congestion Window Size vs Time for Channel Rate of 6 Mbps and Application rate of 10 Mbps

Observations:

- As the application data rate increases, the number of packet drops also increases. This is expected since more packets are being sent due to an increased application data rate.
- The time between decrement of `cwnd` decreases since the congestion is reached earlier due to packets being generated faster.
- For the case where the application rate exceeds the channel rate, we observe a very crowded graph. This is due to the fact that packets are being generated faster than the channel's bandwidth so more packets are being dropped due to that, so the congestion window changes rapidly.

Part 3

Here, I have used the same RateErrorModel for both the NetDevices of N3, i.e. the ones which connect to N1 and N2.

The congestion window size vs time figures are as follows:

- Configuration 1: Figure 15, Figure 16 and Figure 17 show the graphs for Connections 1, 2 and 3 respectively.
- Configuration 2: Figure 18, Figure 19 and Figure 20 show the graphs for Connections 1, 2 and 3 respectively.
- Configuration 3: Figure 21, Figure 22 and Figure 23 show the graphs for Connections 1, 2 and 3 respectively.

The figures also mention the number of dropped packets for each case, and it is also summarised in **??**

Table 4: Number of packet drops for different configurations and connections

Configuration	Connection	Packet Drops
1	1	32
1	2	42
1	3	29
2	1	48
2	2	27
2	3	31
3	1	43
3	2	43
3	3	30

In the congestion avoidance phase, TCP NewReno increases `cwnd` by the `MSS^2 / cwnd` on ack of each segment. This means that increase in `cwnd` decreases as `cwnd` increases, giving it the distinctive curved shape as we see in figures. TCP NewRenoCSE on the other hand increases `cwnd` by `MSS` on ack of each segment, so the increase is the same irrespective of `cwnd`. Thus `cwnd` will increase linearly in congestion avoidance phase.

Observations:

- Between configuration 1 and configuration 2, where the only change is that of connection 3's sender using TCP NewRenoCSE, we observe that the peak `cwnd` reached is noticeably lower in the latter case and the number of packet drops is very marginally higher. The lower peak `cwnd` reaching can explained by how NewReno increases `cwnd` much more slowly than NewRenoCSE and can thus reach to the maximum more closely.
- Between configuration 1 and configuration 3, where all senders use NewReno vs all senders using NewRenoCSE, we similarly observe maximum `cwnd` is lower in the latter than the former. So the network is underutilized with NewRenoCSE.

Impact on the network:

- Having NewRenoCSE on one sender as in the second configuration does not seem to impact the network in any noticeable way. Of course the sender using NewRenoCSE will not be able to reach the max `cwnd` as it could with increasing it slowly at higher rates, the other senders' `cwnd` variation looks to be the same as before when everyone was using NewReno. We can observe that connection 3's max cwnd increases after 25s when the other 2 sender applications have stopped, so it is utilizing available capacity at the sink.

- When everyone uses NewRenoCSE, there are more number of total packet drops since cwnd is increased recklessly even when closer to congestion. It is also increased slower as compared to NewReno after starting from 1, so there is more underutilization of the capacity.

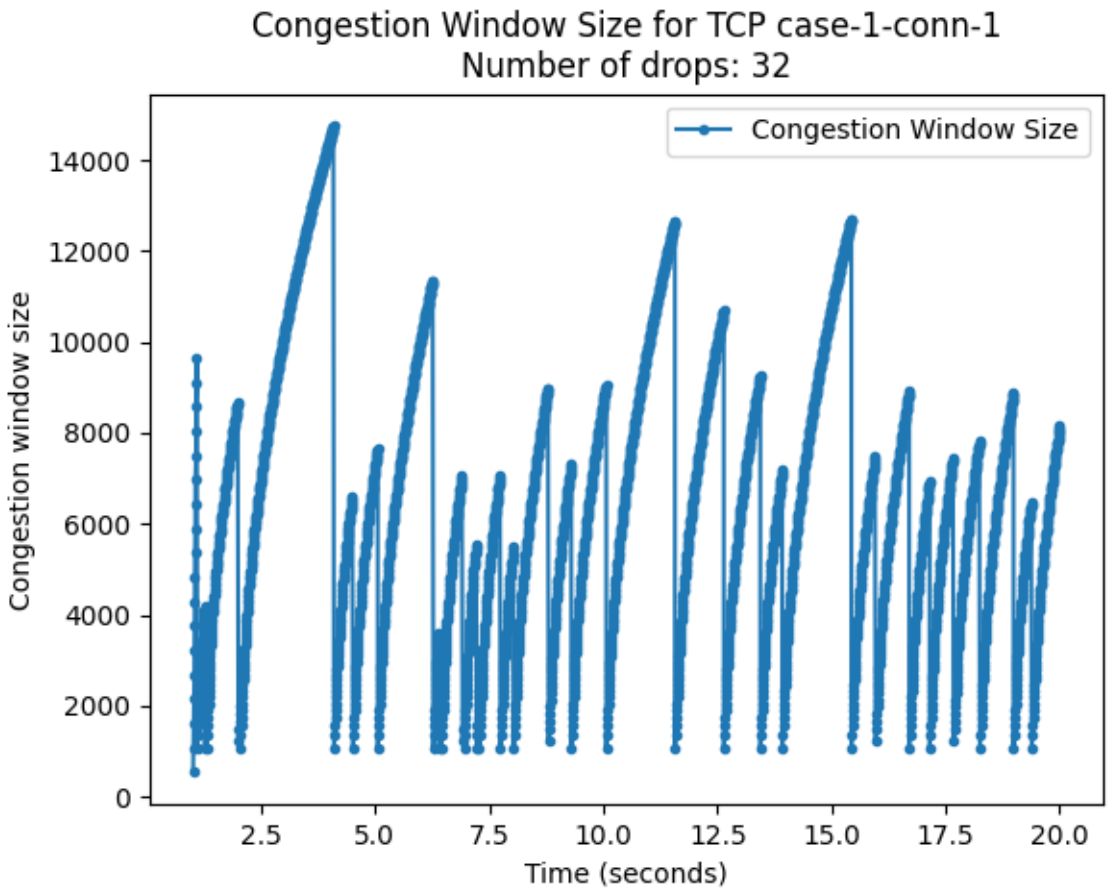

Figure 15: Congestion Window Size vs Time for Connection 1 of Configuration 1

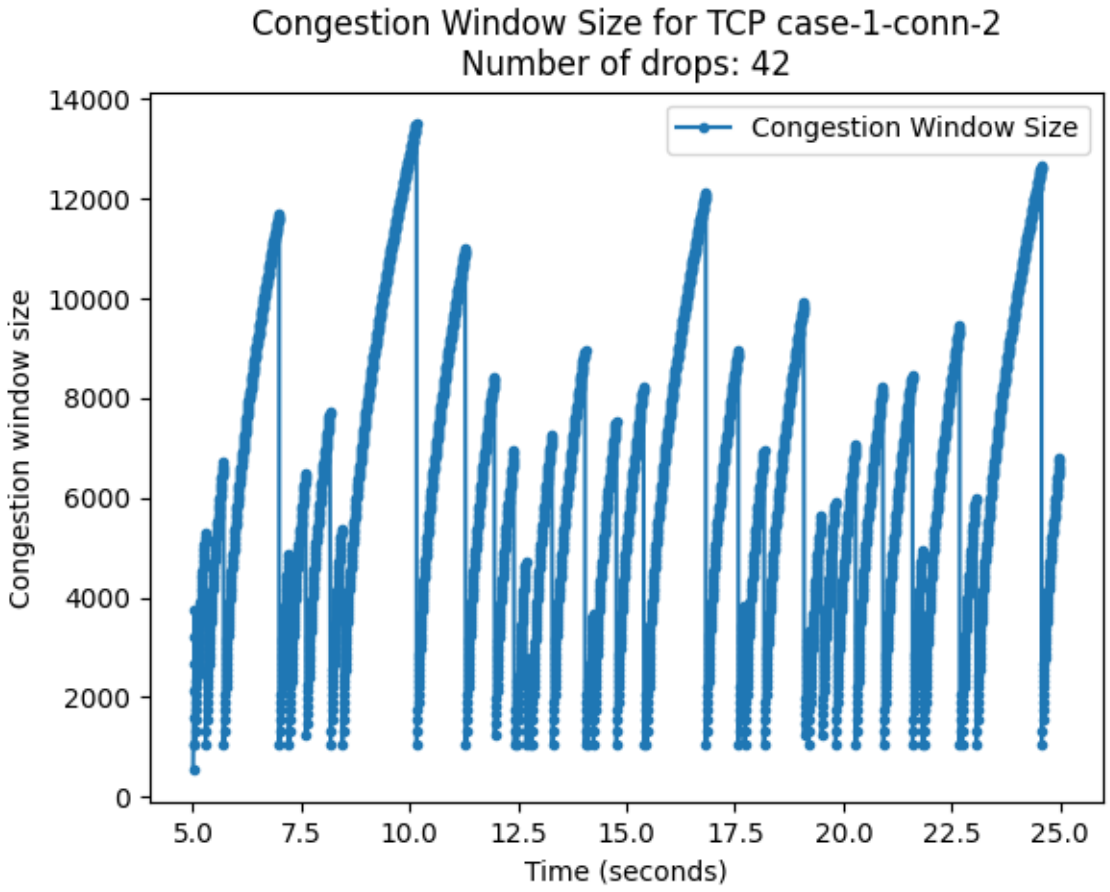

Figure 16: Congestion Window Size vs Time for Connection 2 of Configuration 1

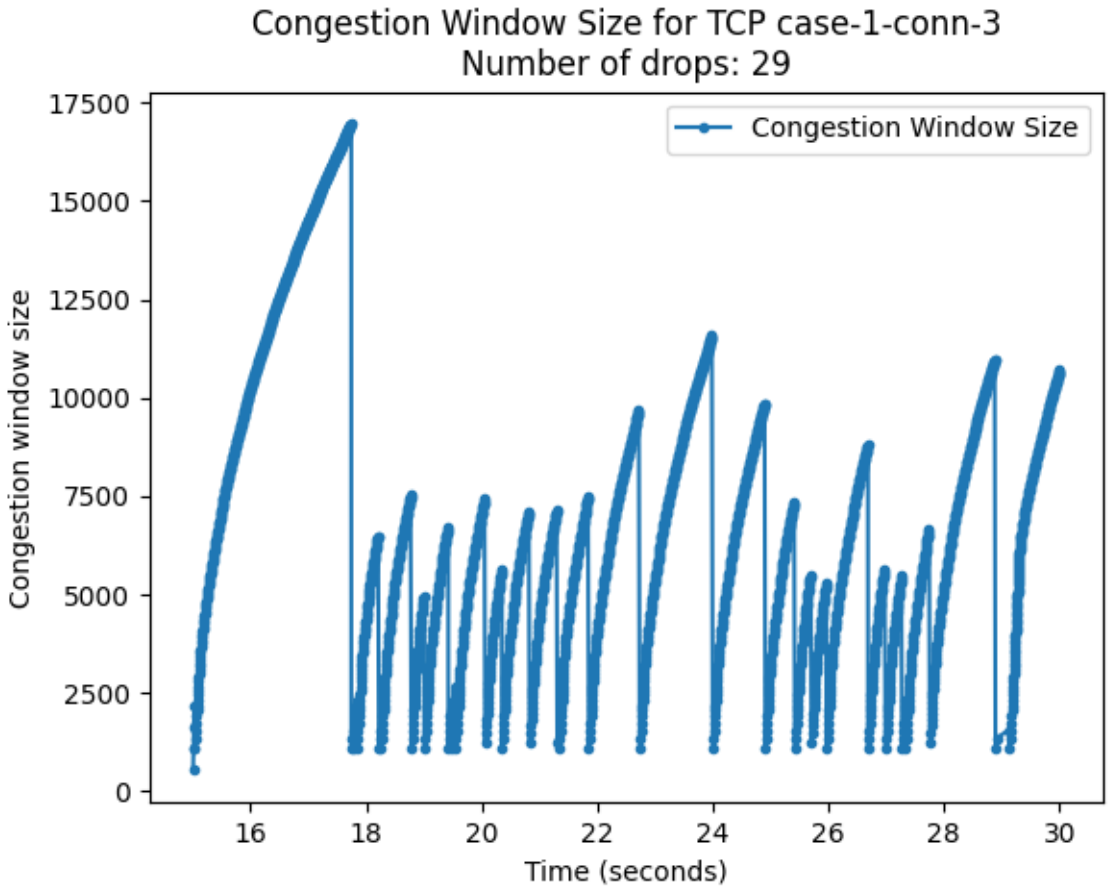

Figure 17: Congestion Window Size vs Time for Connection 3 of Configuration 1

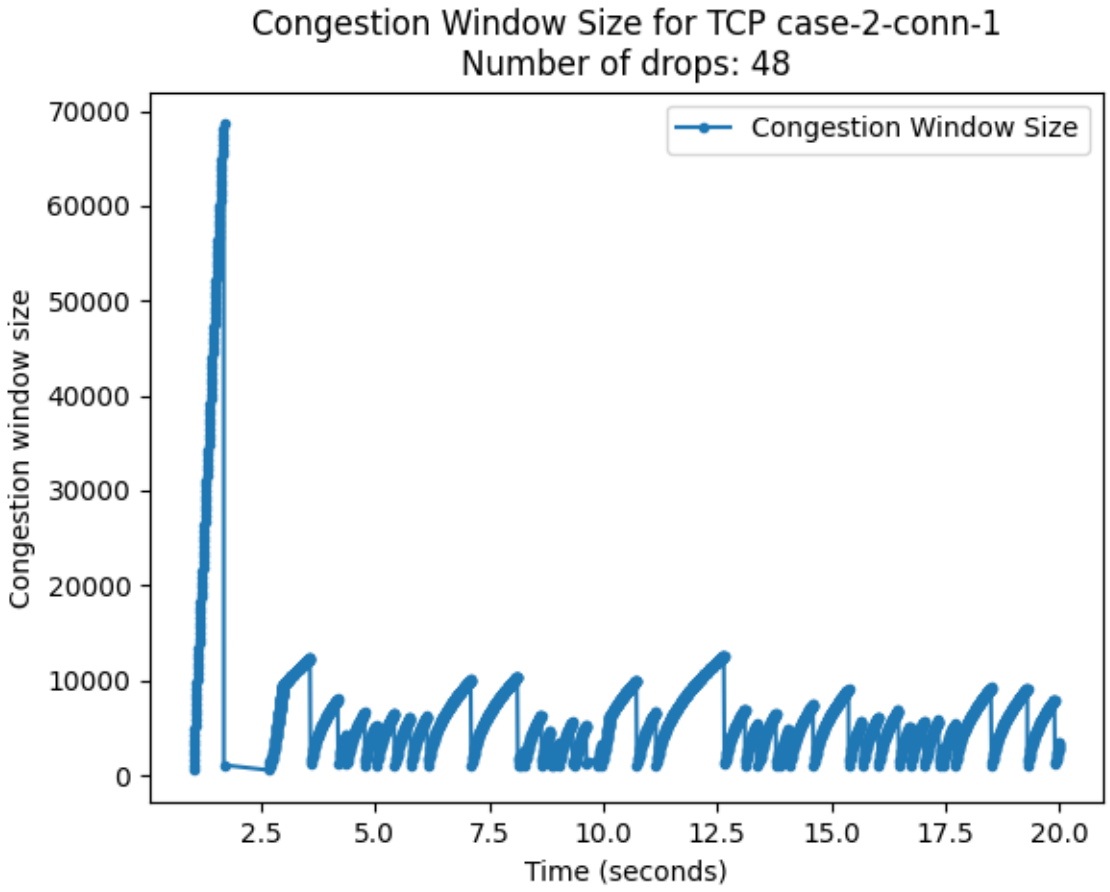

Figure 18: Congestion Window Size vs Time for Connection 1 of Configuration 2

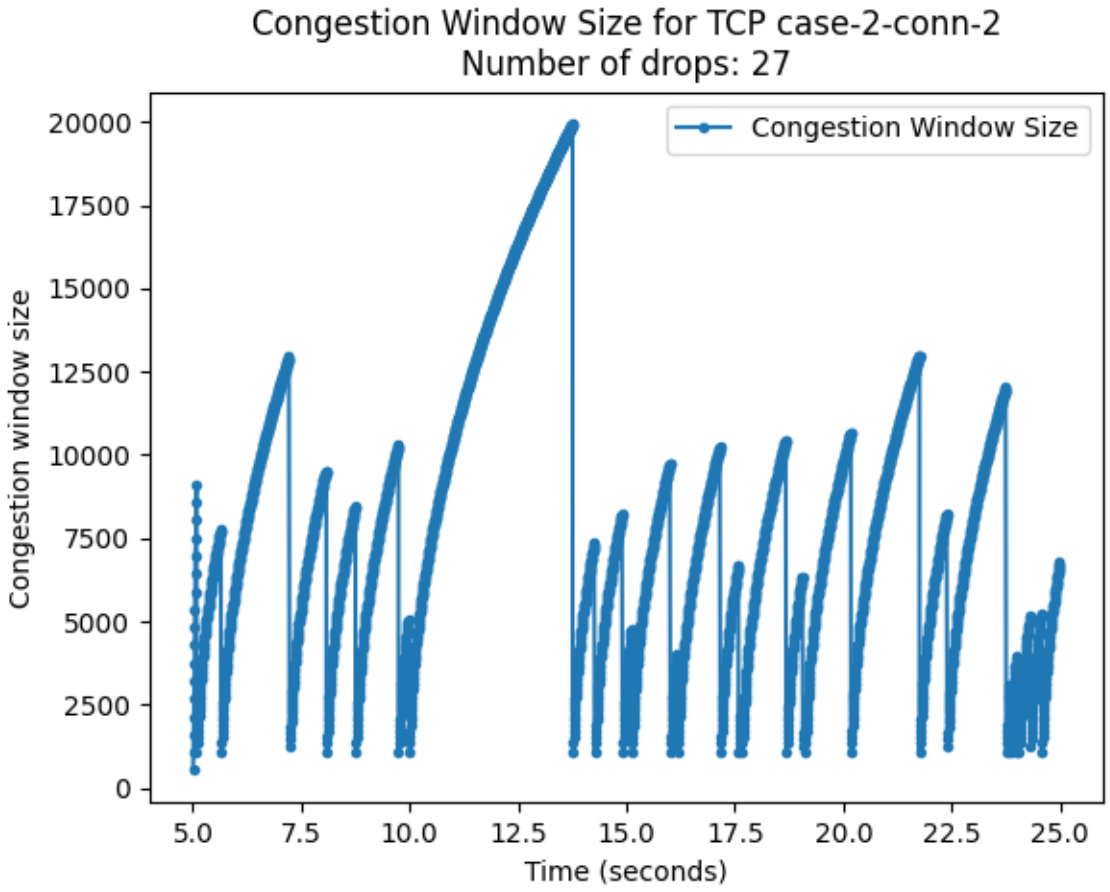

Figure 19: Congestion Window Size vs Time for Connection 2 of Configuration 2

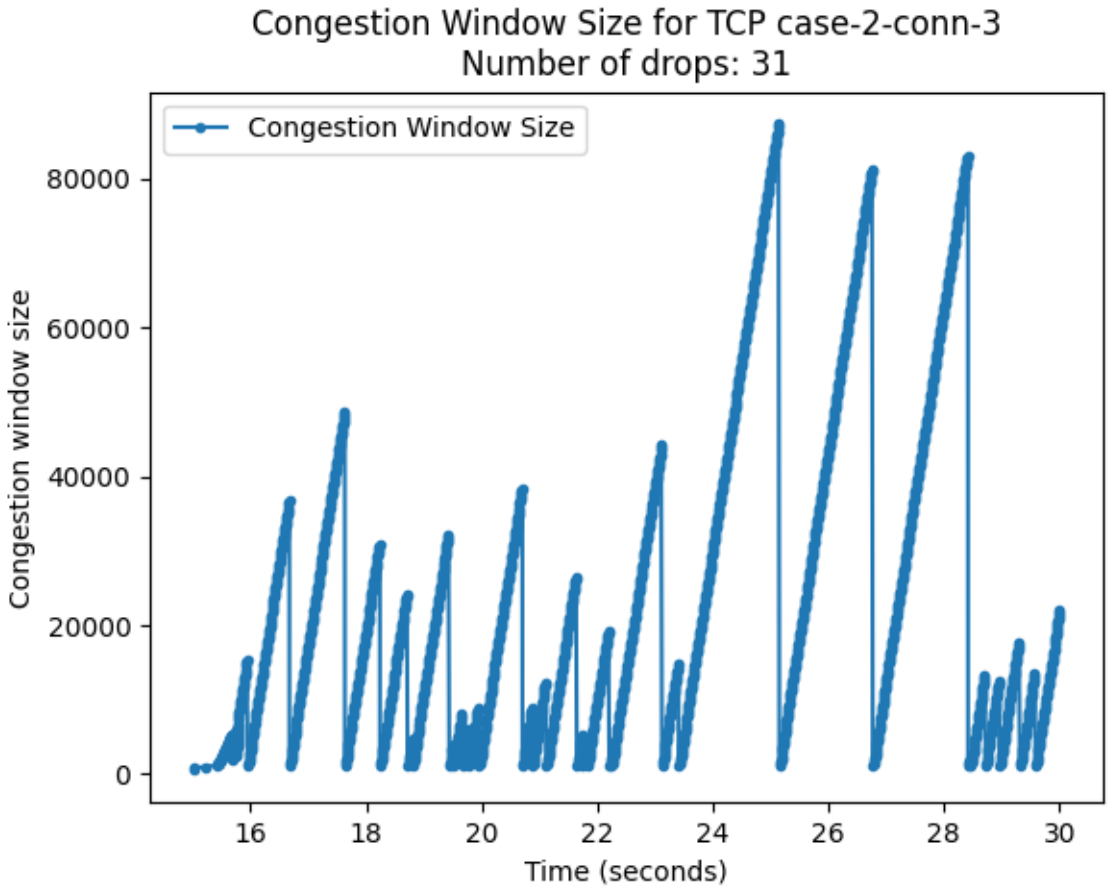

Figure 20: Congestion Window Size vs Time for Connection 3 of Configuration 2

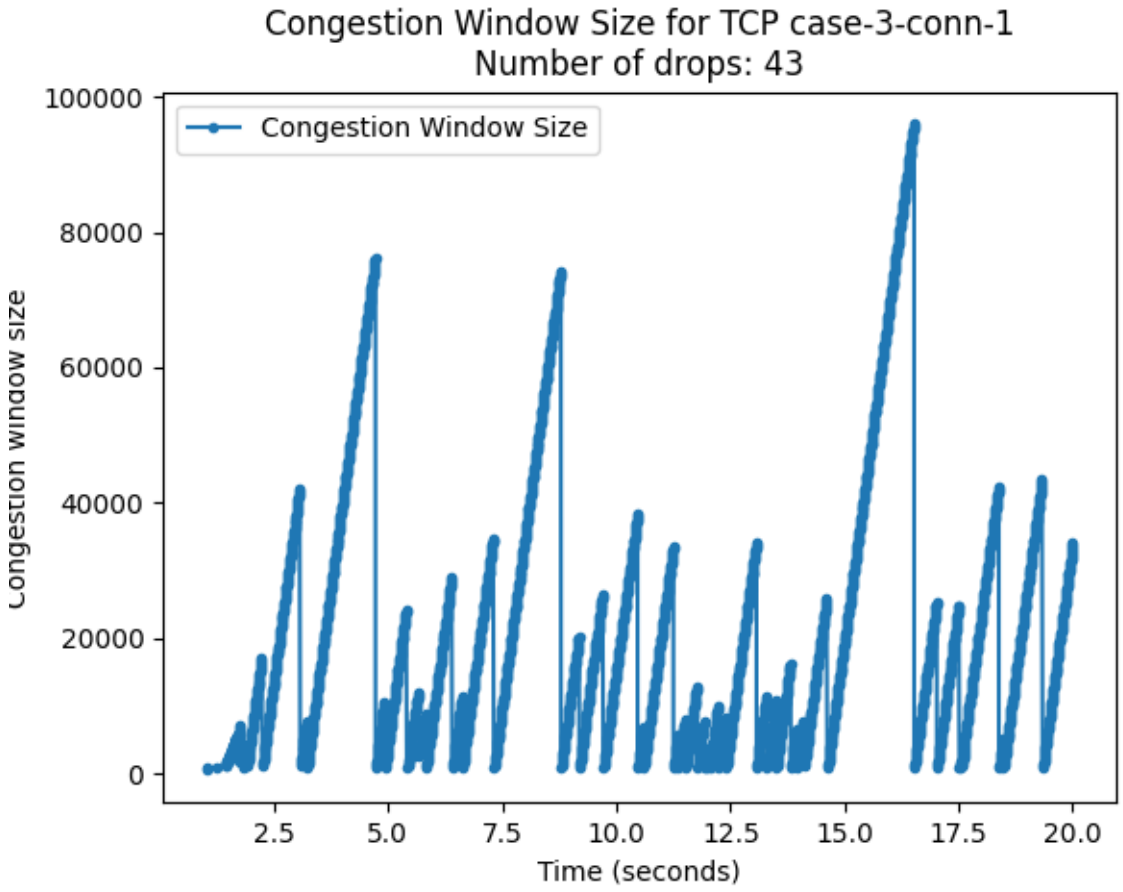

Figure 21: Congestion Window Size vs Time for Connection 1 of Configuration 3

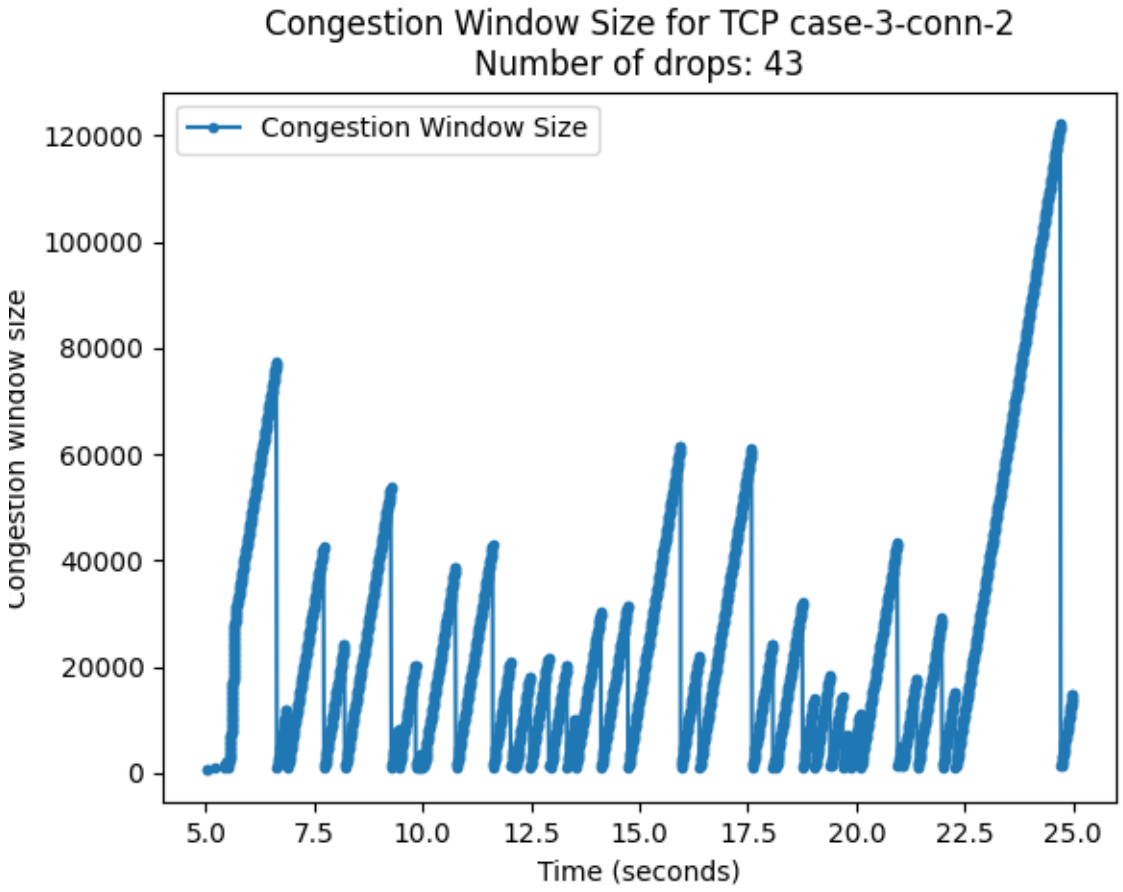

Figure 22: Congestion Window Size vs Time for Connection 2 of Configuration 3

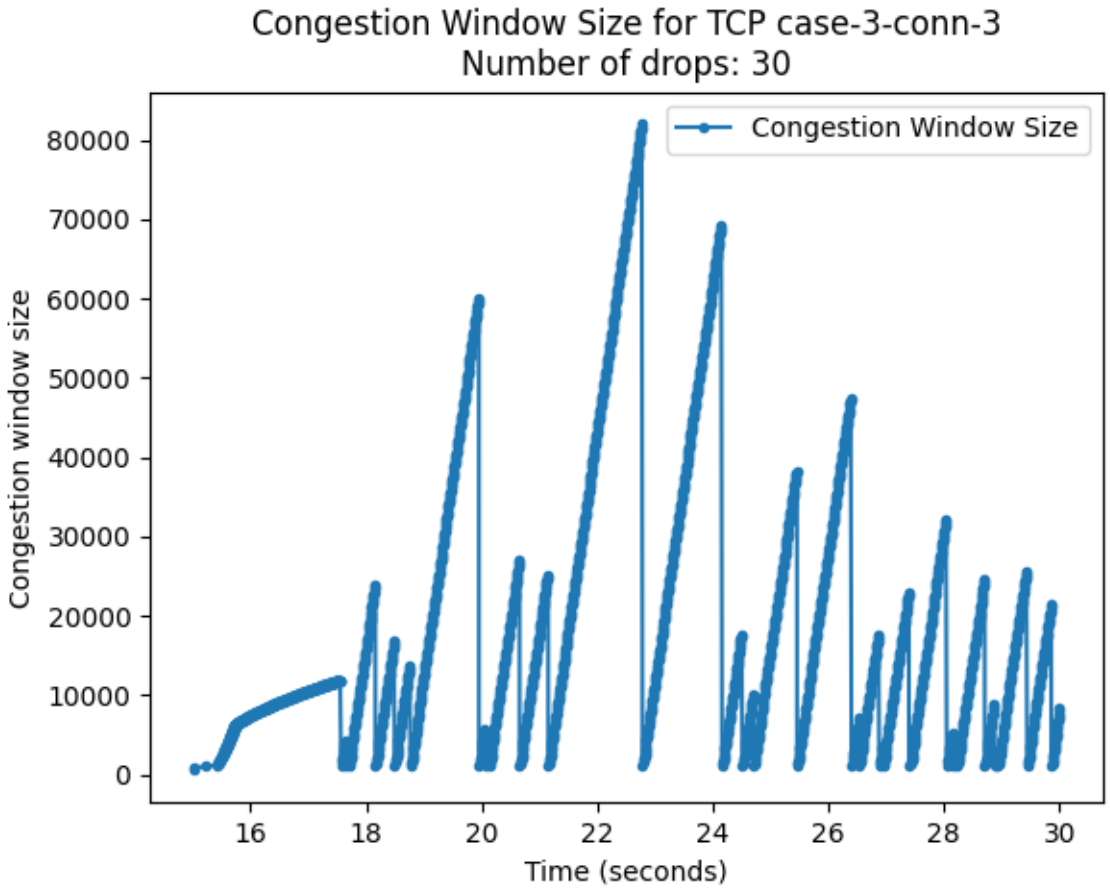

Figure 23: Congestion Window Size vs Time for Connection 3 of Configuration 3