COL334 Assignment 2: Report

Sachin 2019CS10722

October 2021

1 Part 1

1.1 TCP Newreno

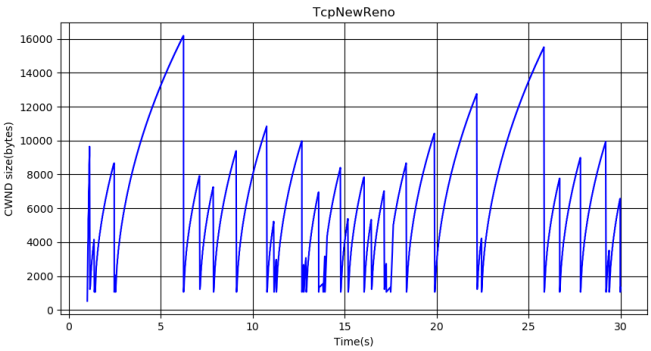

Figure 1: TcpNewReno

So basic idea of Newreno is to increase segment congestion window linearly in slow start state and once congestion window becomes greater than ssThresh and still no congestion is detected, shift to congestion state and increase cwnd by $\frac{segmentSize^2}{cwnd}$, same can be seen from the graph. As the packet is dropped we can see that it takes time to got to high peak, as ssThresh is also decreased.

No of packets dropped = 38

This can be deduced from the graph that number of packets drooped are roughly same as number of peaks in the graph, this is quite reasonable because wehnever the packet is dropped, cwnd is reduced then it start increasing again.

1.2 TCP Veno

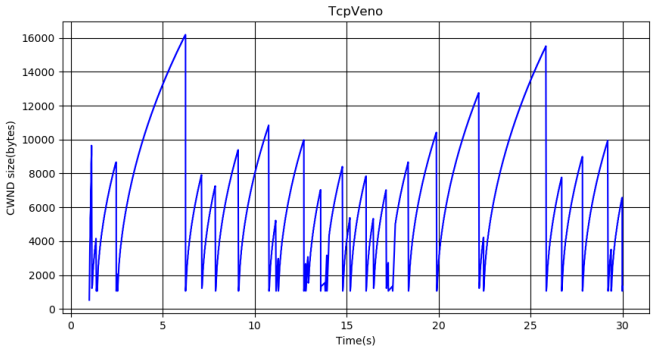

Figure 2: TcpVeno

It can be seen from the graph that Veno's congestion window pattern is very much same as newreno. This is reasonable because Veno is designed upon newreno to deal with random packet loss in wireless network. But as we are simulating simple point to point wired network, it does not do much of the modifications).

No of packets dropped = 38

Again it is equal to packets drop in newreno and roughly equal to number of peaks of graph.

1.3 TCP Highspeed

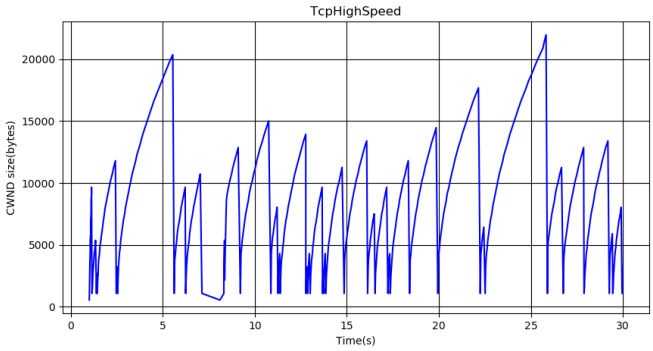

Figure 3: TcpHighSpeed

As stated in the ns3 documentation "TCP HighSpeed is designed for high-capacity channels", it tries to increase cwnd more faster than the standard algorithm (so as to use the window size properly). This is very much evident from the graph as out of all 4 graphs this as the highest peak (more than 20 kB).

No of packets dropped = 38

Packets dropped are same as newreno, which makes it a better protocol than newreno in terms of window usage as from the simulation it is evident that without dropping much packets we are able to increase window size to a larger extent.

1.4 TCP Vegas

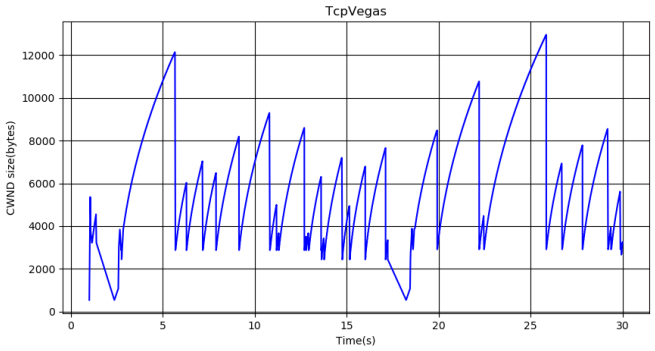

Figure 4: TcpVegas

TCP Vegas maintains small backlog at the bottleneck queue such that it maintains that difference between $\frac{cwnd}{RTT}$ and $\frac{cwnd}{BaseRTT}$ always lies between fixed range. It can be seen from the graph that maximum cwnd attained is low also the times when cwnd goes to 0 is also higher than other 3. This is because it also limits cwnd from upper side (difference is bounded)

No of packets dropped = 39

Number of packets dropped is slightly larger than other 3 protocols.

2 Part 2

2.1 Varrying Channel Data Rate

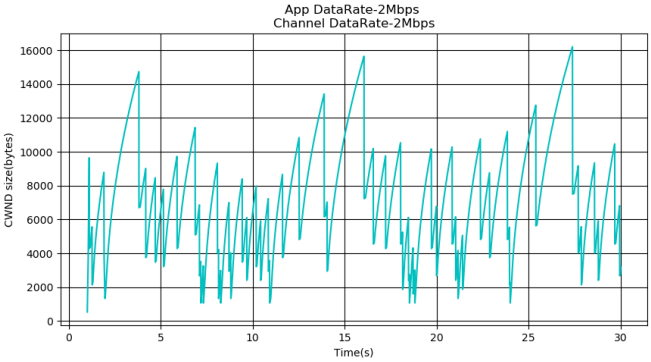

Figure 5: Channel Data Rate = 2Mbps

No of packets dropped = 62

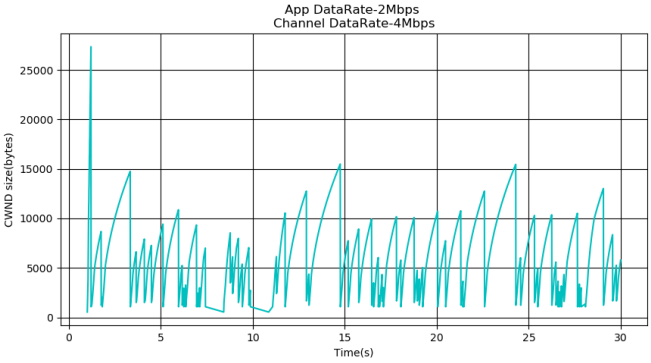

Figure 6: Channel Data Rate = 4Mbps

No of packets dropped = 72

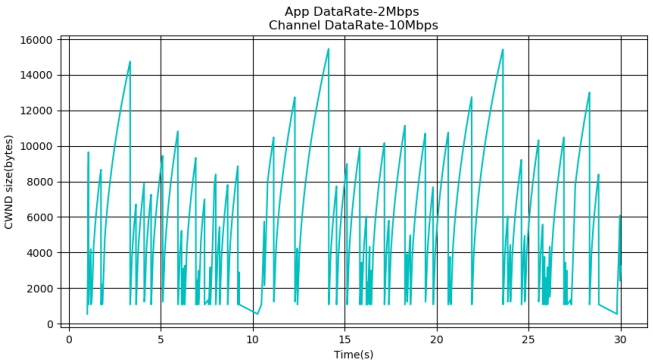

Figure 7: Channel Data Rate = 10Mbps

No of packets dropped = 73

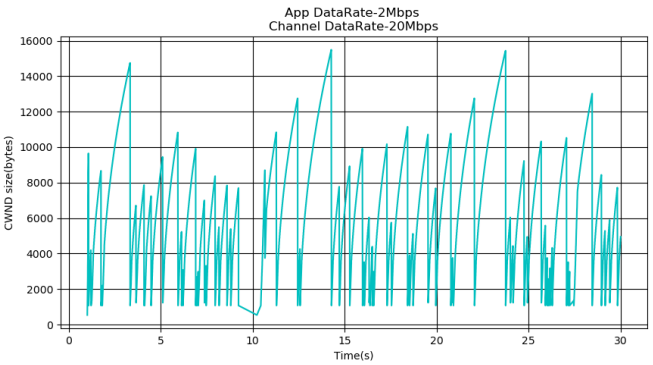

Figure 8: Channel Data Rate = 20Mbps

No of packets dropped = 74

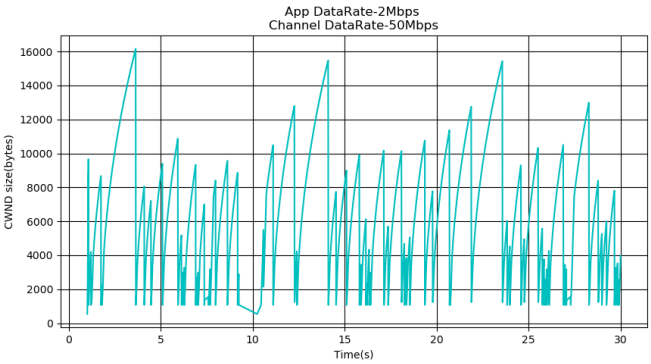

Figure 9: Channel Data Rate = 50Mbps

No of packets dropped = 75

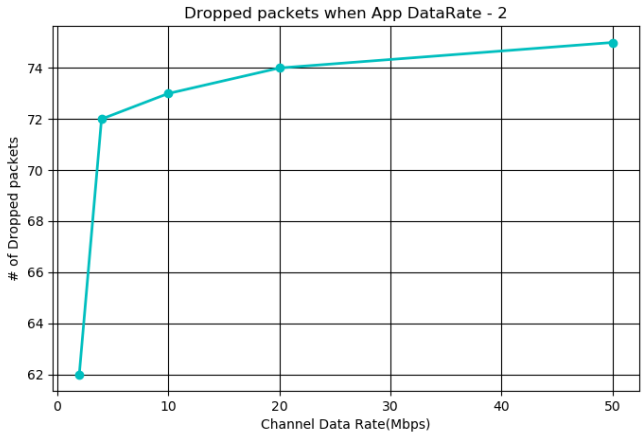

Figure 10: Dropped packets

Increasing channel data rate causes data to travel faster inside the channel, so RTT will decrease and we will get ack faster. The number of packets dropped will have increasing behaviour.

2.2 Varrying Application Data Rate

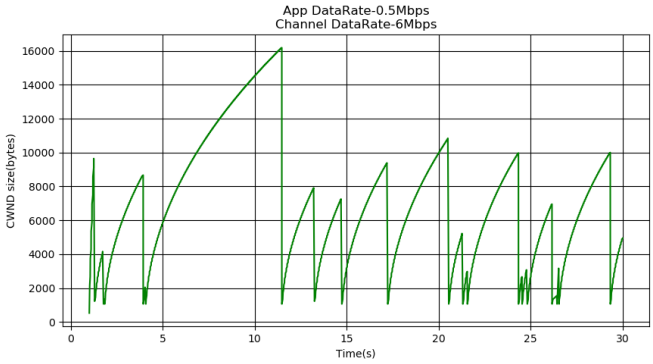

Figure 11: Application Data Rate = 0.5Mbps

No of packets dropped = 22

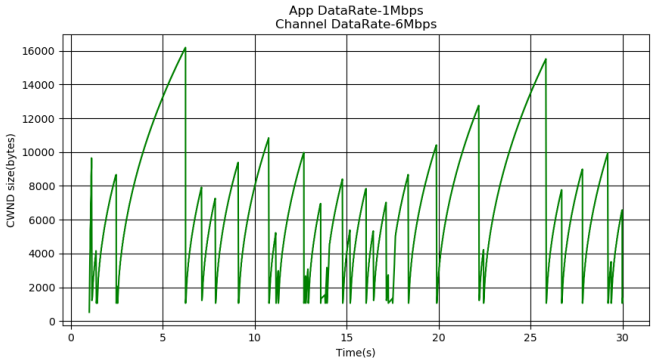

Figure 12: Application Data Rate = 1Mbps

No of packets dropped = 38

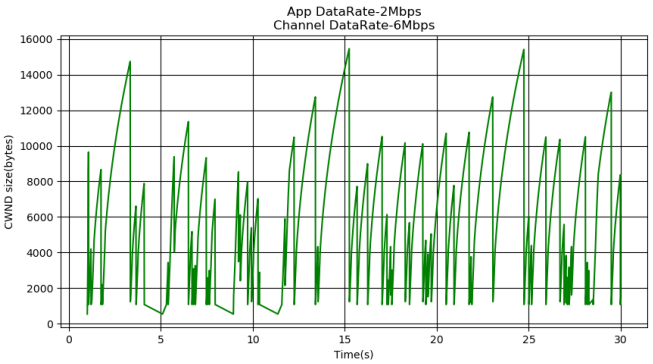

Figure 13: Application Data Rate = 2Mbps

No of packets dropped = 71

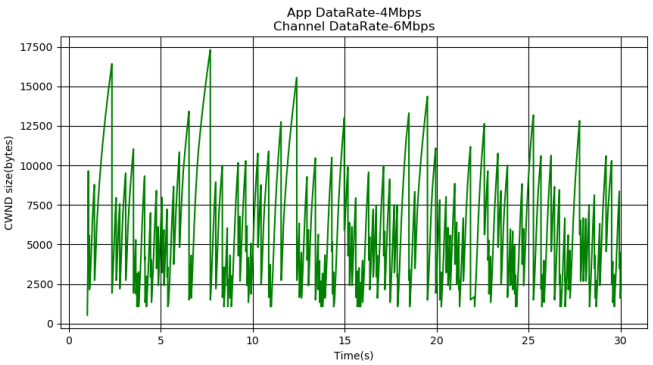

Figure 14: Application Data Rate = 4Mbps

No of packets dropped = 156

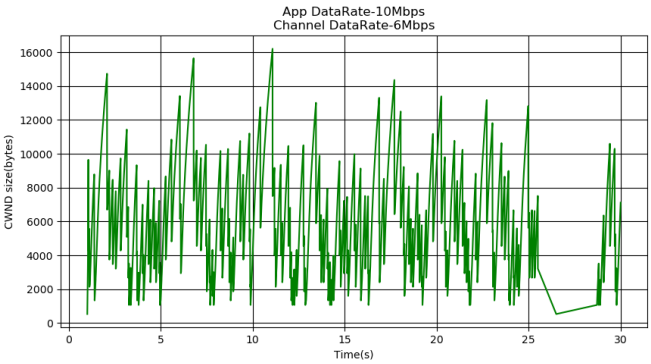

Figure 15: Application Data Rate = 10Mbps

No of packets dropped = 156

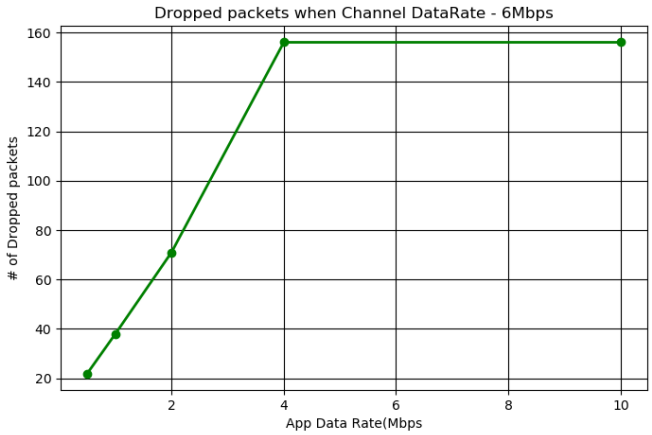

Figure 16: Dropped packets

Increasing application data rate causes data to arrive at transport layer of TCP of source faster now, since TCP won't send data higher more than cwnd, and also buffer memory is limited so higher the app data rate higher is number of packets dropped.
Also it is evident from the graph that as app data rate increases, the graph sort of shriks, i.e. now cwnd will tend to change faster (increase faster, decrease faster).

3 Part 3

The new TCP protocol designed is very much same as TCp newreno, just the behaviour of slow start stae and congestion avoidence states are reversed with some changes in constatnts. As the behaviour of slow start and congestion avoidence is set to be that way with the reason to avoid congestion i.e. to decrease window size whenever network is congestion prone, but since this new protocol reverses both the roles, so now window size is likely to be high when congestion is high inside network and low when congestion is low. Also our congestion avoidence state increases cwnd linearly, graph is more likely to linear in the areas where it was exponnetial for newreno.

3.1 Configuration 1

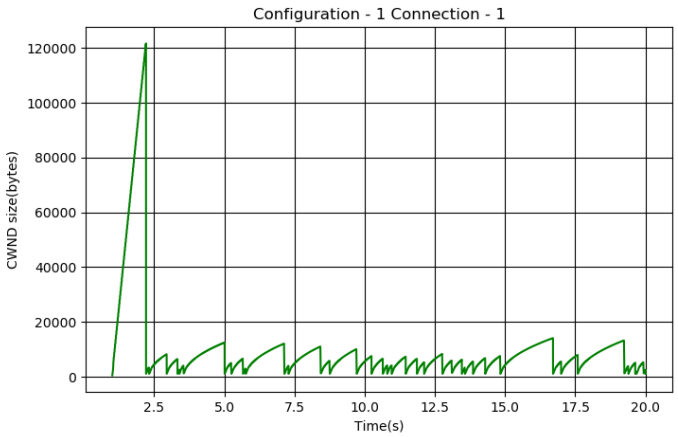

Figure 17: Connection 1

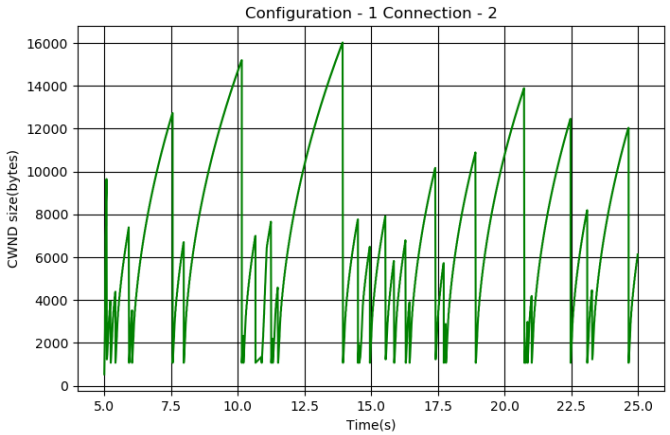

Figure 18: Connection 2

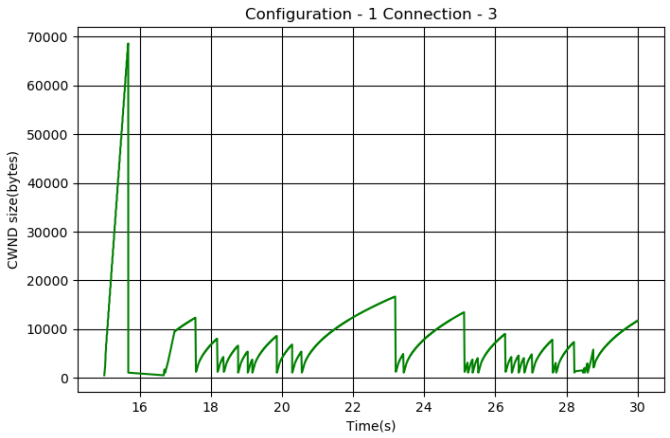

Figure 19: Connection 3

No of packets dropped = 109

Initially when there was only 1 connection active, there was less congestion inside network hence for connection 1 initial peak is higher.
As soon as second connection gets established congestion increases. And when all 3 are present peaks are low.

3.2 Configuration 2

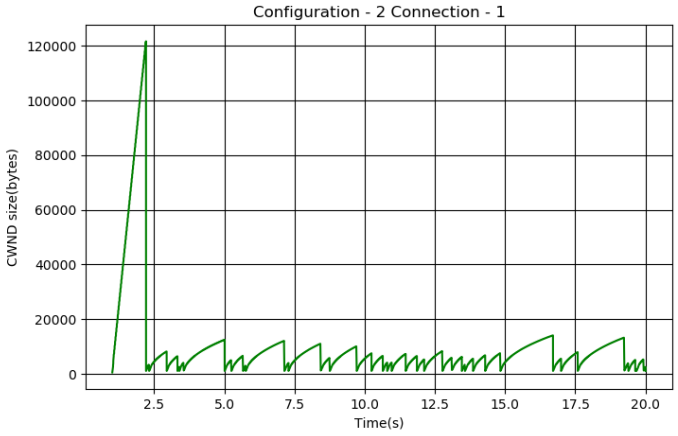

Figure 20: Connection 1

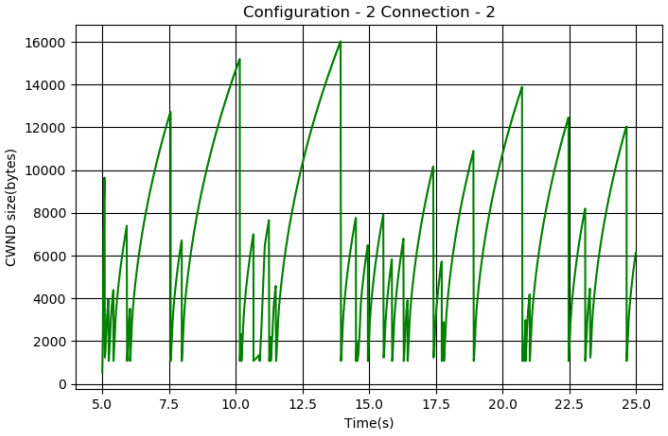

Figure 21: Connection 2

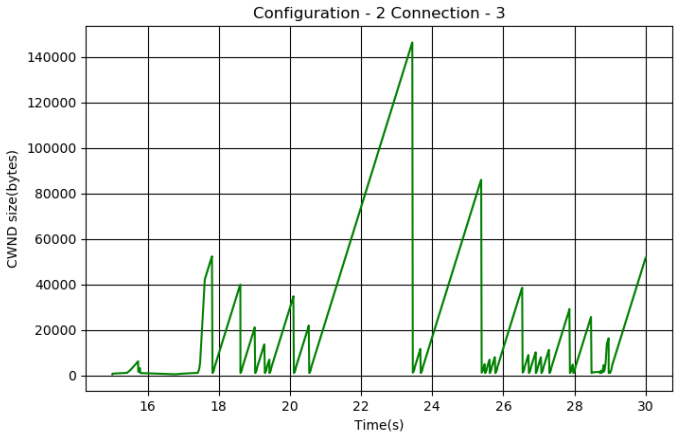

Figure 22: Connection 3

No of packets dropped = 106

Now since 3rd connection is newreno CSE, it will start from low congestion window and tend to increase its window size when congestion is high, this can be easily seen from graph. And it have linear nature in congestion avoidence state.

3.3 Configuration 3

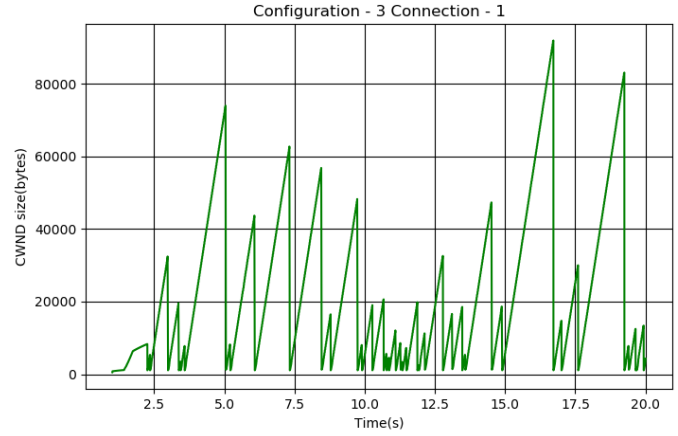

Figure 23: Connection 1

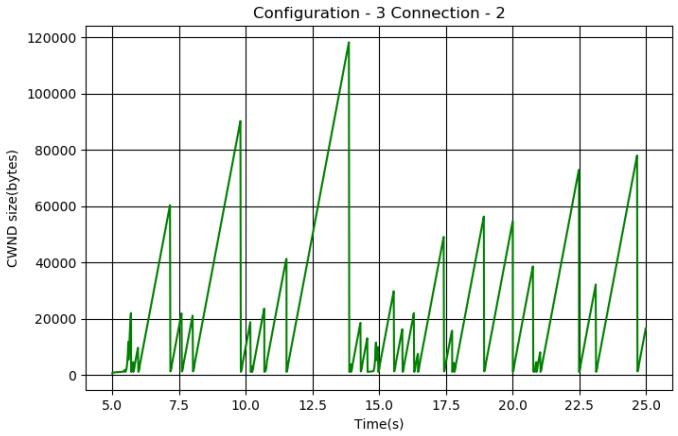

Figure 24: Connection 2

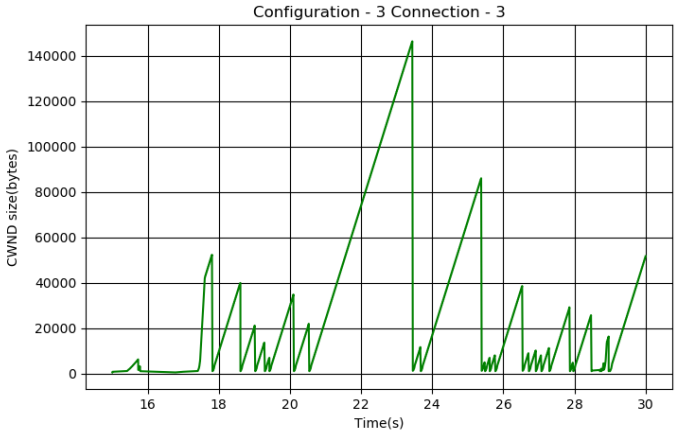

Figure 25: Connection 3

No of packets dropped = 110

Now that all the connections are newreno CSE, congestion window of all follows the behaviour stated in the start of this question. Also peak values are higher. Round the time when all 3 connections are on, peaks are higher.

4 How to Run

The file inside folder with their usage are as follows:

1. **First.cc:** Code for part a, present inside folder Q1
2. **Second.cc:** Code for part b, present inside folder Q2
3. **Third.cc:** Code for part c, present inside folder Q3
4. **TcpNewRenoCSE.h:** Header for the new protocol required for part c, present inside folder Q3/Congestion
5. **TcpNewRenoCSE.cc:** Code for the new protocol required for part c, present inside folder Q3/Congestion
6. **plots.py.cc:** Helper file to plot the graphs by reading output files present inside ns3.29/ folder (NOTE: this code only plots the graph for file that are already saved, not runs the actual ./waf command)
7. **runner.py:** Helper file to run different parts of assignment.

Steps to run the actual code:-

1. Unzip the file
2. Put files 'First.cc', 'Second.cc', 'Third.cc' inside ns3.29/scratch folder
3. Put files 'runner.py' and 'plots.py' inside ns3.29/ folder (You can run them from anywhere but the you have to give the path of there files, since we will be running all commands from ns3.29 only its better to put them there.)
4. Put file 'TcpNewRenoCSE.cc' and 'TcpNewRenoCSE.h' inside folder ns3.29/src/internet/model
5. Build and compile the ns3 program
6. Run runner.py with command [python3 runner.py], commands for all 3 parts will start running.
7. if you want to run each part by using ./waf yourself, follow the following constraint:-
   (a) For First.cc run the command
   [./waf –run "scratch/First -TCP=arg"] where arg is the TCP protocol you want to run eg 'TcpNewReno', 'TcpVeno' etc.

(b) For Second.cc run the command
[./waf –run " scratch/Second -ADR=arg1 -CDR=arg2] where arg1 is Application data rate and arg2 is Channel Data Rate.

(c) For Third.cc run the command [./waf –run " scratch/Third -C=arg] where arg is configuration type (1,2,3).

8. You can set -P=False while running any part to avoid printing output to console.