COL334 Assignment-3

Shrey J. Patel, 2019CS10400

October 2021

1 Submission Contents

All the submitted files are included a top-level directory called **2019CS10400**.

- **Q1/-**
  1. **Outputs_1**:
     - <***Name of protocol***>***.cwnd*** for congestion window vs time data
     - ***PacketDrop_***<***Name of protocol***>***.txt*** for packet drop data
  2. **Plots_1**:
     - <***Name of protocol***>***.png*** for congestion vs time plot
  3. **First.cc**: The actual C++ code for the first part.
- **Q2/-**
  1. **Outputs_2**:
     - <***TypeOfDataRate***>-<***ValueOfDataRate***>***.cwnd*** for congestion window vs time data
  2. **Plots_2**:
     - <***TypeOfDataRate***>-<***ValueOfDataRate***>***.png*** for congestion vs time plot
  3. **Second.cc**: The actual C++ code for the second part.
- **Q3/-**
  1. **Congestion:**
     The header and body codes for the custom congestion protocol **TcpNewRenoCSE**.
  2. **Outputs_3**:
     - <***ConfigurationNo***>-<***ConnectionNo***>***.cwnd*** for congestion window vs time data
     - ***NetPacketDrop_***<***ConfigurationNo***> for the total number of packets dropped in given configuration
     - ***PacketDrop_***<***ConfigurationNo***>-<***LinkNo***> for number of packets dropped in given configuration and link
  3. **Plots_3**:

- ***<ConfigurationNo>-<ConnectionNo>.png*** for congestion window vs time plot

4. **Third.cc**: The actual C++ code for the third part.

- **plot_cwnd.py**: A plotting script which takes in the output files as input and returns a Congestion Window Size v/s Time plot.

*Note: Before plotting in any of the parts, put **plot_cwnd.py** in the **ns3.29** directory.*

2 Part1: Analyzing different congestion protocols

2.1 Execution instructions:

- Put the script **First.cc** in the path ***ns3.29/scratch/Q1***.
- To run the code for part 1, type:

  ./waf –run "scratch/Q1/Q1 –tcp_type –analysis_type"

  Where the argument **tcp_type** takes in a string corresponding to the desired TCP congestion protocol out of the following:

  1. TcpNewReno
  2. TcpHighSpeed
  3. TcpVeno
  4. TcpVegas

  and the argument **analysis_type** takes two values:

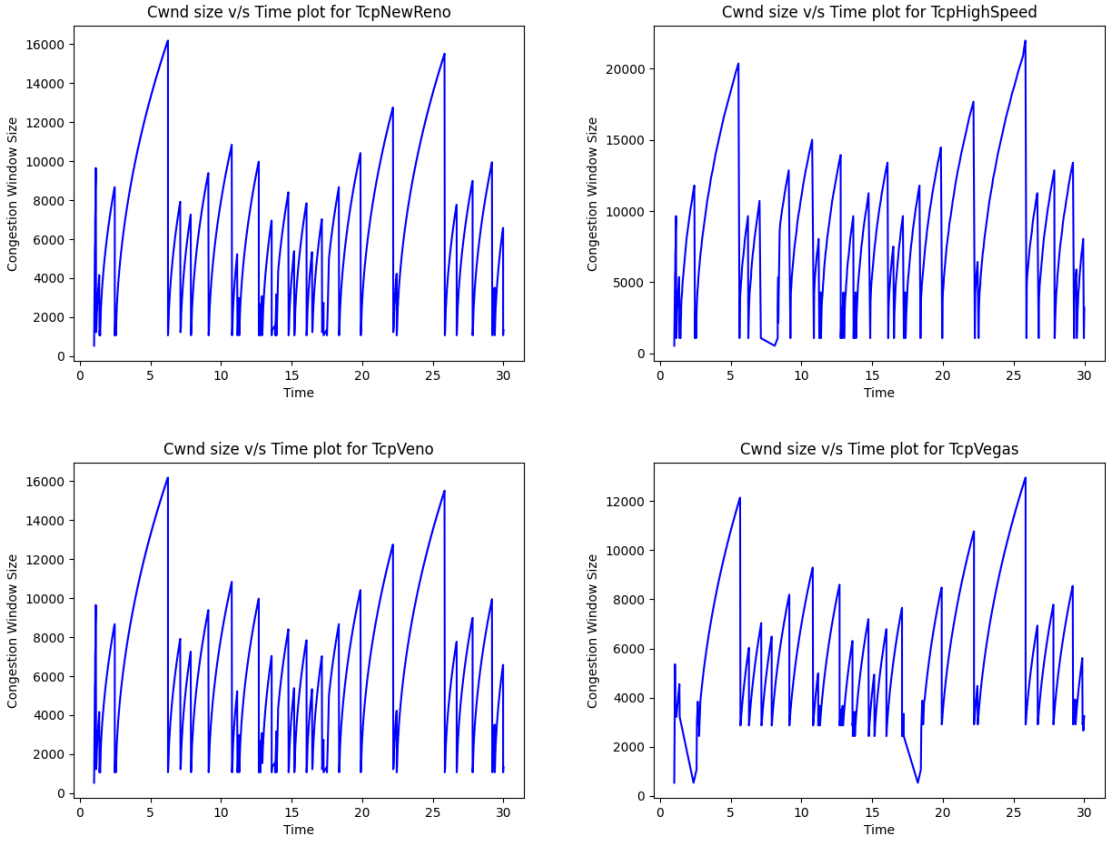
  - 0: Analysis of congestion window size
  - 1: Analysis of packet drop

- For plotting,

  python3 plot_cwnd.py Outputs_1/<FileName>

  the plots will be saved in the folder Plots_1 with the same naming conventions

2.2 Plots



2.3 Packet Drops

Number of packets dropped in TcpNewReno = 38
Number of packets dropped in TcpHighSpeed = 38
Number of packets dropped in TcpVeno = 38
Number of packets dropped in TcpVegas = 39

Inference:
The packets are dropped whenever the congestion window size exceeds the maximum cwnd size allowed by the protocol depending on various different factors like congestion in the intermediate network, size of the receiver buffer, etc. All the buffers have a coping mechanism in case of packet loss, and the most evident mechanism is to decrease the congestion window size, so as to send lesser number of segments and thus decrease the chances of further congestion. Since before the packet drop, the congestion window size is continuously increasing, packet loss due to any congestion event is observable from the graphs as abrupt peaks. *Thus, the number of packet drops in a given time period can be obtained from the number of peaks in the graph in that given time interval.*

And this mechanism more or less holds regardless of the type of congestion protocol. The things that differ in different protocols are the rate of increase in cwnd sizes during slow start and congestion avoidance, the congestion threshold, rate of decrease after packet loss, etc. All this factors only affect the values of the maximum congestion window, while the actual trend of packet losses remains same. This can be seen in the plots of different congestion protocols, in which the only difference in the Y-axis values, while the actual number of peaks are the same, resulting in almost same number of packet drops.

2.4 Description

All the observations and explanations are based on the plots, the official implementations and documentation of the TCP protocols.

2.4.1 TcpNewReno

- In the slow start phase, the congestion window size increases as:

  $$\textbf{cwnd += SMSS(sender maximum segment size)}$$

  for every acknowledged segment. However, later versions of NewReno also accommodate unacknowledged segments by introducing N(number of previously unacknowledged segments) to the slow start phase.

- The increase in cwnd size is roughly constant with time in this phase and thus, the slow start phase is evident in the initial ascending part of the plots.

- The slow start phase ends when the cwnd size reaches/exceeds the threshold, after which the protocol reaches the congestion avoidance phase in which cwnd increases as follows:

  $$\textbf{cwnd += max(1,}(SMSS)^2\textbf{/cwnd)}$$

  From the formula, cwnd size increases by at atleast 1 per RTT, and the increase in size is inversely proportional to the current size of the cwnd.

- Thus, the rate of increase is negative in the congestion avoidance phase, which is evident in the concave nature of the graph near the congestion.

- After a congestion event occurs, the threshold for slow start phase is halved, which can be observed from the graph as the peaks are decreasing with consecutive packet drops.

2.4.2 TcpHighSpeed

- This protocol differs from other ones in not having any slow start phase i.e. the cwnd size increases by the same formula till the congestion(thus, it only has congestion avoidance phase):

  $$\textbf{cwnd += a(cwnd)/cwnd}$$

  where a(cwnd) is obtained from a lookup table depending upon the current value of congestion window. As argued before, the rate of increase decreases with time because of cwnd in denominator.

- Presence of a slow start phase, which is sometimes rapid and independent of the current cwnd size, causes congestion and absence of this phase in HighSpeed makes it suitable for using in networks in heavy congestion without many packet losses, since it starts in the congestion avoidance phase itself.
- In the event of a packet drop, the threshold also changes according to the function:

  cwnd = cwnd(1 - b(cwnd))

  where b(cwnd) is also obtained from the lookup table. Thus, the multiplicative decrease also depends upon current congestion window size, and thus this makes TcpHighSpeed suitable for high capacity channels since all packets drops in this case will not result in the same reduction factor, and this gives us a chance to keep the congestion window size just near the maximum allowable size.
- The high capacity is visible from the plots since HighSpeed has the highest peak values of all the other congestion protocols.

2.4.3 TcpVeno

- After looking at the actual implementation of the code of Veno, I found that the slow start and congestion avoidance methods of TcpVeno call the corresponding methods of NewReno.
- This is the reason why the data values and the plots of Veno are almost identical to that of NewReno.
- However, according to official documentation, Veno actually enhances NewReno by implemetation more effective packet loss strategies, since it implements some properties of Vegas as well. But, for the observable time interval, the plots for both are same(except in a small interval near t=12s).

2.4.4 TcpVegas

- This protocol is a delay based protocol, as opposed to others which are drop based(except Veno). So it tries to prevent drops instead of trying to detect them by maintaining a small backlog.
- It continuously samples the RTT values to calculate the actual throughtput and compares it with the expected throughput. The difference between them provides an estimate of backlogged packets.

  Actual Throughtput = cwnd/sampled RTT
  Expected Throuhput = cwnd/baseRTT
  diff = Expected - Actual

- The increase/decrease in cwnd sizes are linear and dependant on the fact whether the number of backlogged packets are kept between certain bounds, so as to ensure maximum calculated throughtput. In other words, TcpVegas is more cautious due to which it tries to predict maximum cwnd size through delays instead of actual packet losses.

- This is the main reason why Vegas seems to have the least value of congestion window sizes as compared to other protocols which use drop based coping mechanism.

3 Part2: Analyzing effect of bandwidth/application data rate

3.1 Execution instructions:

- Put the script **Second.cc** in the path ***ns3.29/scratch/Q2***.
- To run the code for part 2, type:

  ./waf –run "scratch/Q2/Q2 –type –value"

  Where the argument **type** takes as input the type of data rate provided

  - 0: Channel Data Rate
  - 1: Application Data Rate

  and the argument **value** takes the corresponding value of the data rate.

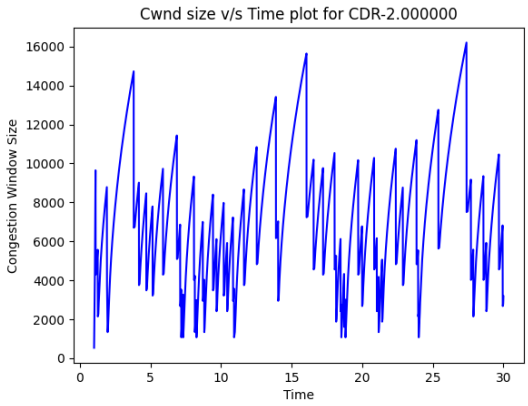
- For plotting,

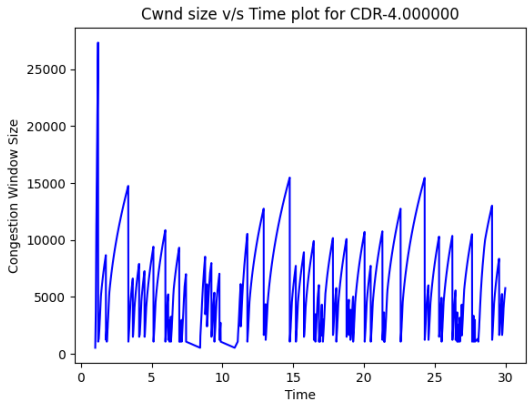
  python3 plot_cwnd.py Outputs_2/<FileName>

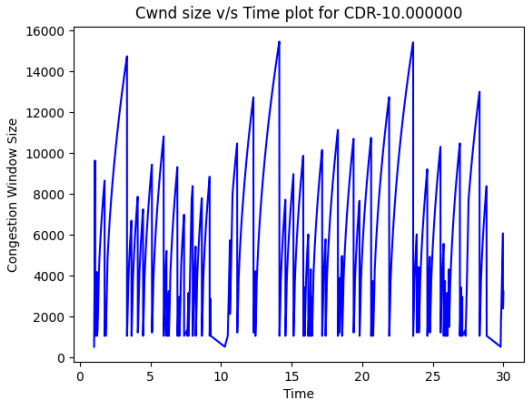
  the plots will be saved in the folder Plots_2 with the same naming conventions

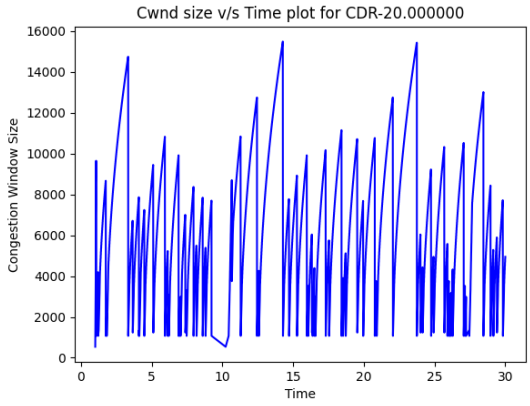
3.2 Subpart (a): Channel Data Rate

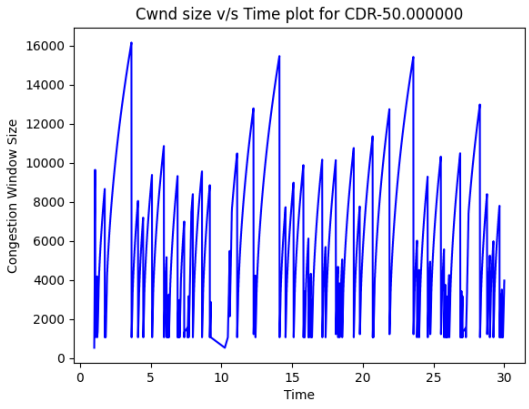
3.2.1 Plots









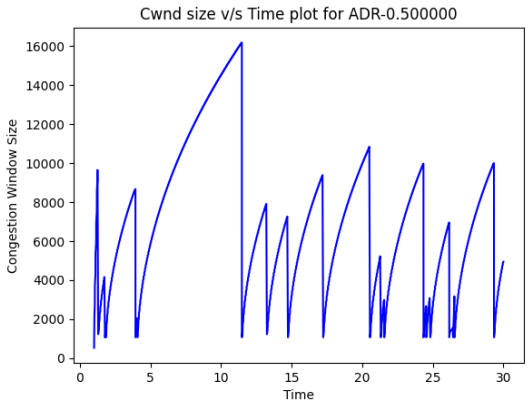


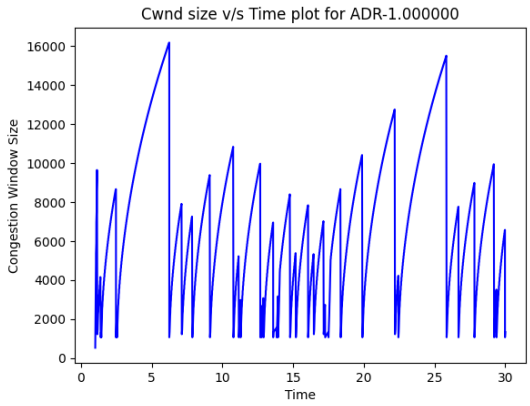
3.2.2 Observations

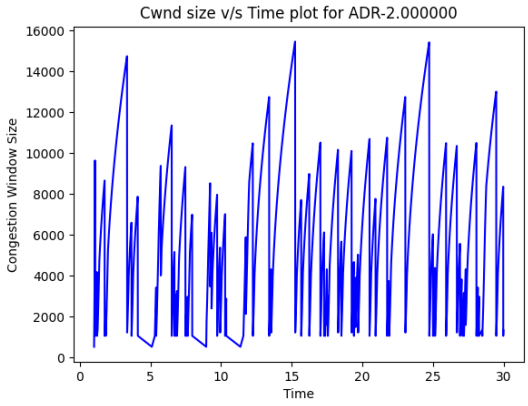
- It is evident that as the channel data rate increases, it becomes possible to transmit more number of segments through the link with time, which is observable from the first three plots, as the frequency of drops keeps on increasing from graphs 1 to 3.
- However, note that the last three plots are almost identical, or in other words, there is no perceptible change in the frequency of congestion window size changes. This may be explained from the fixed application data rate. If the channel bandwidth exceeds the amount which is required by a particular fixed application data rate, the remaining channel bandwidth should in theory remain unused.
- Thus, the graph should not change beyond a certain channel data rate value, since the excess is going to be wasted anyway.

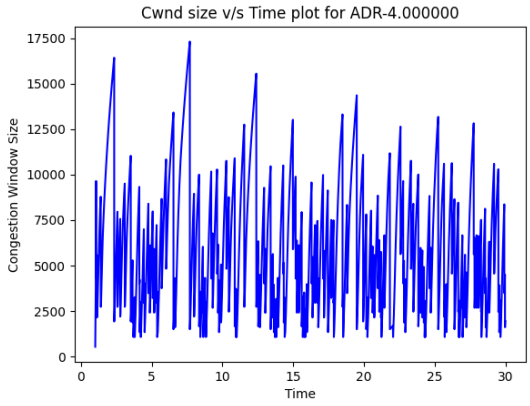
3.3 Subpart (b): Application Data Rate

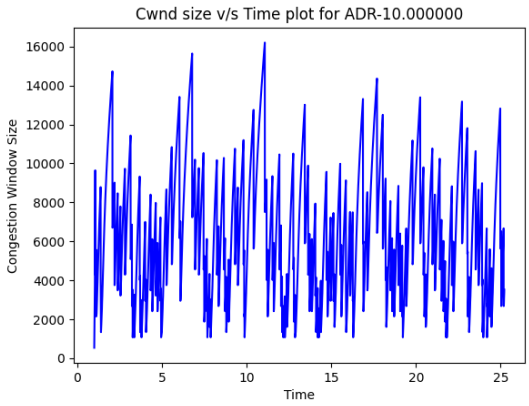
3.3.1 Plots











3.3.2 Observations

- Unlike the channel data rate, the increase in variations in the congestion window size due to increasing application data rates is clearly visible since the packet transmission frequency actually depends on the sender and his application data rate instead of the link rate.
- More transmission rate also means more congestion events and more packet drops. And as explained in part 1, packet drops are related with the peaks of the graph. Thus, the peak frequency also increases with increasing application data rates.

4 Part3: Implementing and Analyzing a custom congestion protocol

4.1 Execution instructions:

- Put the script **Third.cc** in the path ***ns3.29/scratch/Q3***.
- To run the code for part 2, type:

  ./waf –run "scratch/Q3/Q3 –config –type"

  Where the argument **config** takes the configuration number as mentioned in the problem statement and the argument **type** takes in the type of analysis to be done as follows:

  - 0: Analysis of congestion window size
  - 1: Analysis of packet drop
- For plotting,

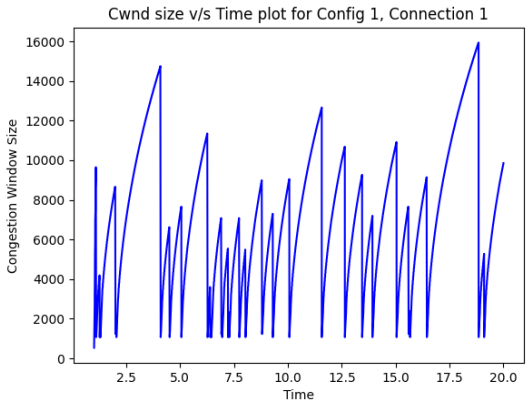
  python3 plot_cwnd.py Outputs_3/<FileName>

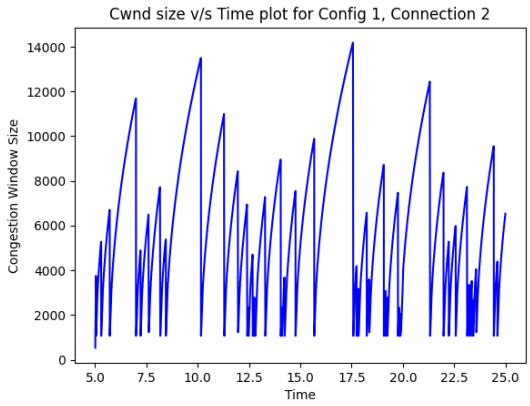
  the plots will be saved in the folder Plots_3 with the same naming conventions

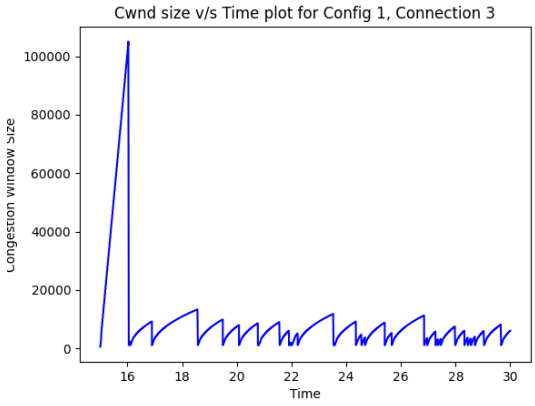
I have used two Rate Error Models on the node N3, one for each point to point link, so as to separately study the packet drop for each link.

4.2 Configuration 1

4.2.1 Plots







4.2.2 Packet Drop

Total number of dropped packets for link 1: 79
Total number of dropped packets for link 2: 34
Total number of dropped packets: 113

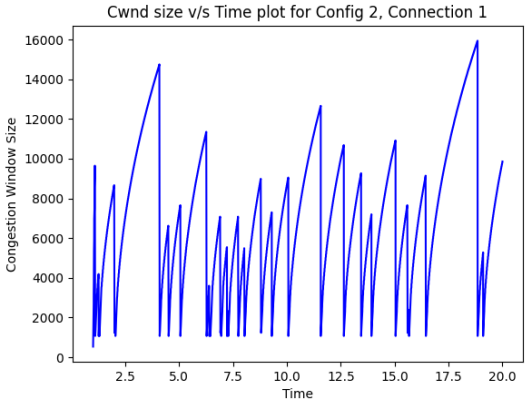
4.2.3 Observations

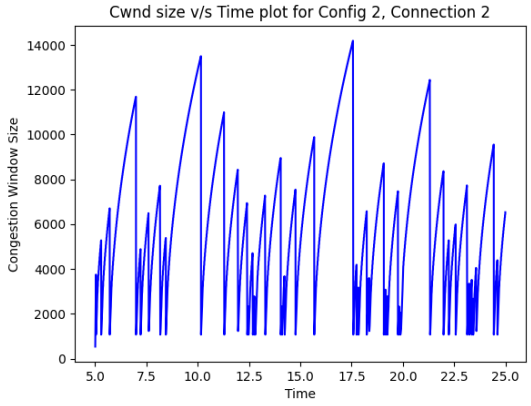
- In this configuration, all the connections use TcpNewReno. So, atleast for the first two connections, the plot is very similar to that obtained in the first part of the assignment. Also, the first two connections have independent sources and sinks but the same link.
- However, the trend is unusual for the third connection, which I can't explain because since the third connection is completely independant from the first two connection with respect to

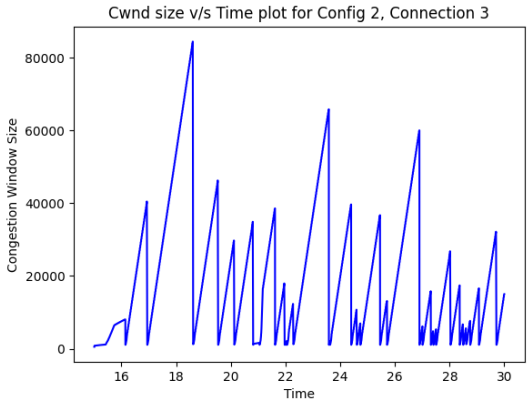
sources, sinks as well as links, the congestion sizes in the first two connections shouldn't affect those in connection 3.

4.3 Configuration 2

4.3.1 Plots







4.3.2 Packet Drop

Total number of dropped packets for link 1: 79
Total number of dropped packets for link 2: 33
Total number of dropped packets: 112

4.3.3 Observations

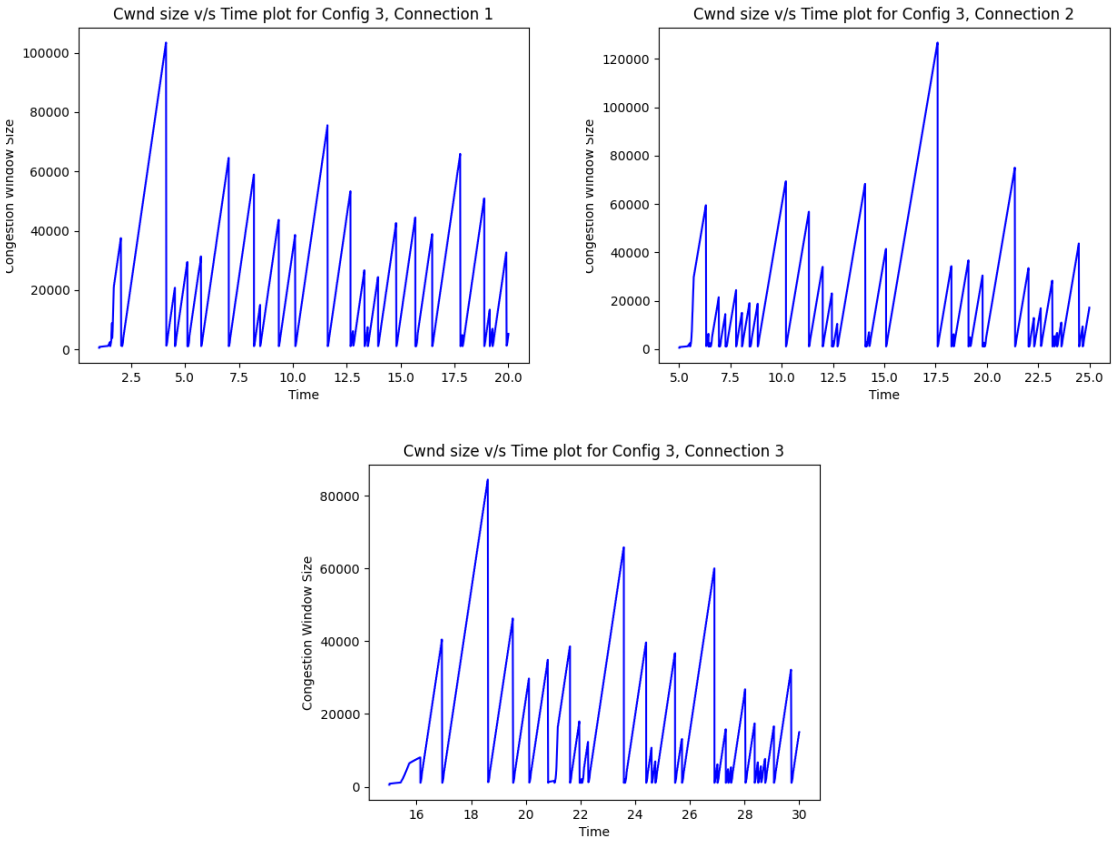
- In this configuration, The first two connections still use TcpNewReno while the third connection uses TcpNewRenoCSE but since they are completely independent of the third connection, the

data and plots for connection 1 and connection 2 are exactly identical as in configuration 1.

- However, since the third connection has a different protocol, its trend is completely different from the one in configuration. However, the most prominent observation is that the peak congestion window sizes in TcpNewRenoCSE are much higher as compared to previous NewRenoCSE.

4.4 Configuration 3

4.4.1 Plots



4.4.2 Packet Drop

Total number of dropped packets for link 1: 77
Total number of dropped packets for link 2: 33
Total number of dropped packets: 110

4.4.3 Observations

In this configuration, we use TCPNewRenoCSE in all the connections and the change is very apparent since both the average and peak congestion window sizes are higher then those in TcpNewReno.

Also note that the slow start and congestion avoidance heuristics of TcpNewRenoCSE are reversed, so the congestion avoidance is no longer inversely proportional to the current cwnd value. Due to which the increase, even near the congestion is linear, which is quite clear from the graph and this results in higher congestion window values.

But, this may result in higher number of packet drops since the avoidance phase is not very cautious as compared to other protocols.