Computer Networks - Assignment 3

Name : Ujjwal Mehta | Entry No. : 2020CS10401

Task 1:

In this part we simulated a 2 node network topology in order to check the variation of number of packet drops and size of the congestion window while varying between different TCP protocols.

Code Explanation :

The code for this part can be found in **task1.cc** file and in order to create this topology I have referred to the code from example file of ns3 from **seventh.cc** file and modied it to count the number of packet drops and and output the time stamps along with the congestion window size in a **task1.txt** file. Upon running this **task1.cc** file in ns3, we will get the number of packet drops in terminal output as well as a time and congestion window size data in **task1.txt**. In order to obtain the number of packet drops, I have maintained a global int variable named **COUNT_DROP** whose value is incremented every time the packet drop is found (i.e. everytime the RxDrop function is called) and finally I print the variable on the terminal when the simulation stops. Once we get the **task1.txt** file, I use the **task1_plot.py** script which I created to find the maximum congestion window size and print it in terminal and form the plot of congestion window with time using **Matplotlib Library in Python**.

Finally in order to change between different Tcp protocols, I used the following statement to configure the default Tcp protocols:

```
Config::SetDefault ("ns3::TcpL4Protocol::SocketType", StringValue
("ns3::TcpWestwood"));
```

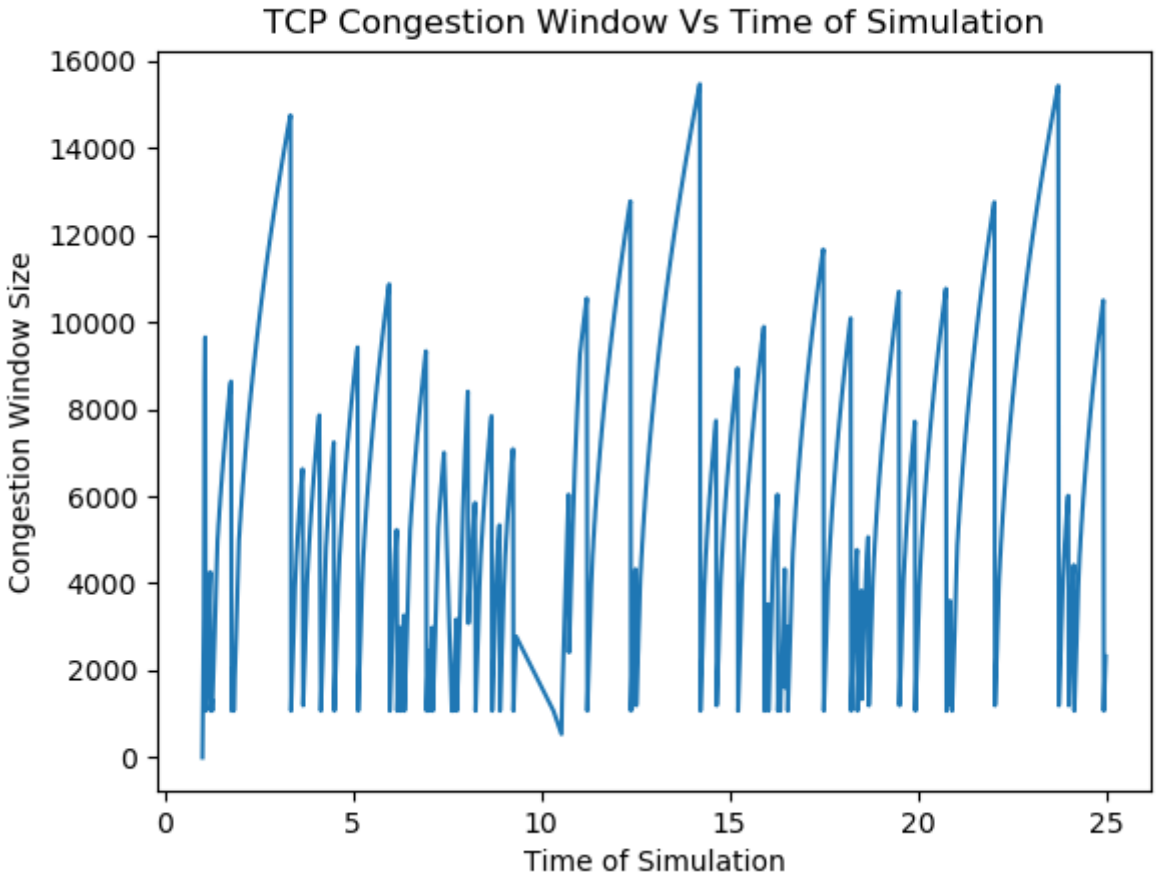
Just change the name of protocol to use it.

1. Results :

Tcp Protocol	Maximum Cogestion Window Size	Number of Packet Drops
New Reno	15462	58
Vegas	11252	58
Veno	15462	59
WestWood	15471	60

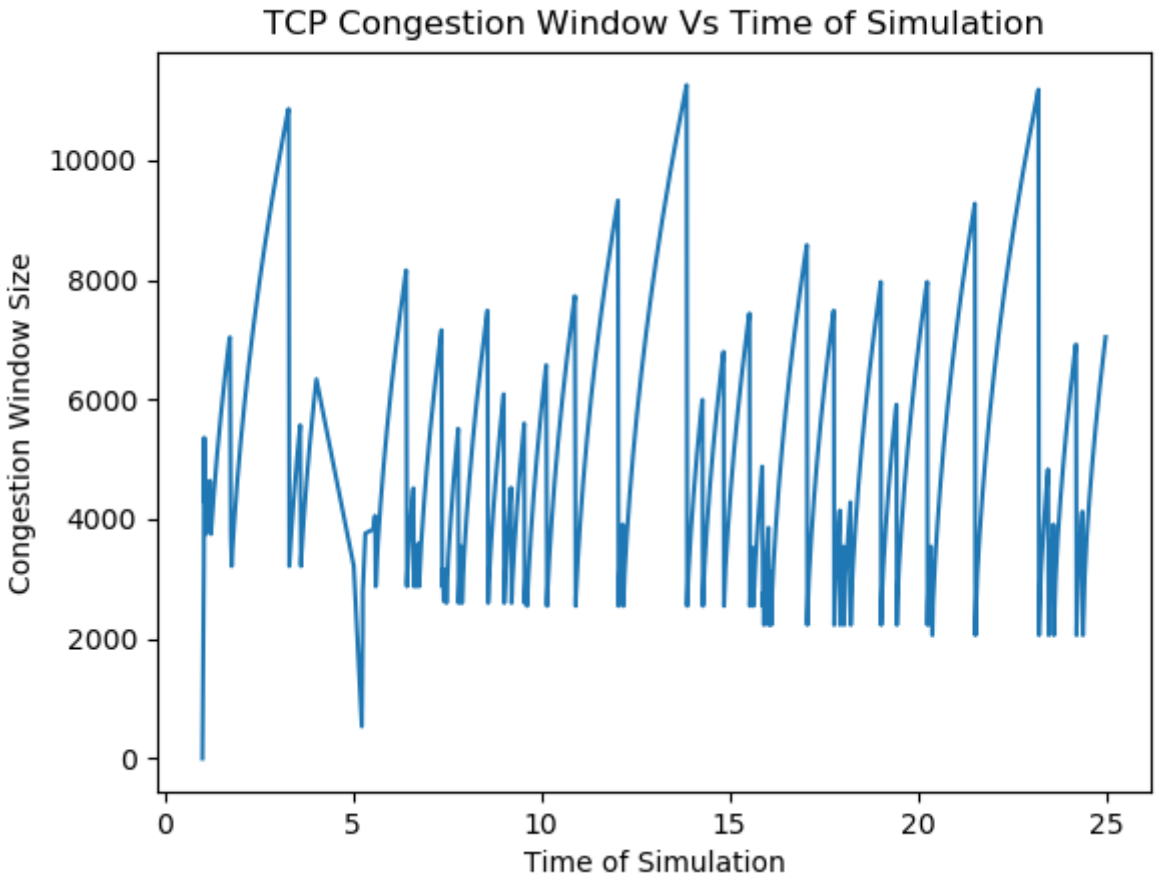
2. Graphs of Congestion Window Vs Time :

Tcp New Reno:



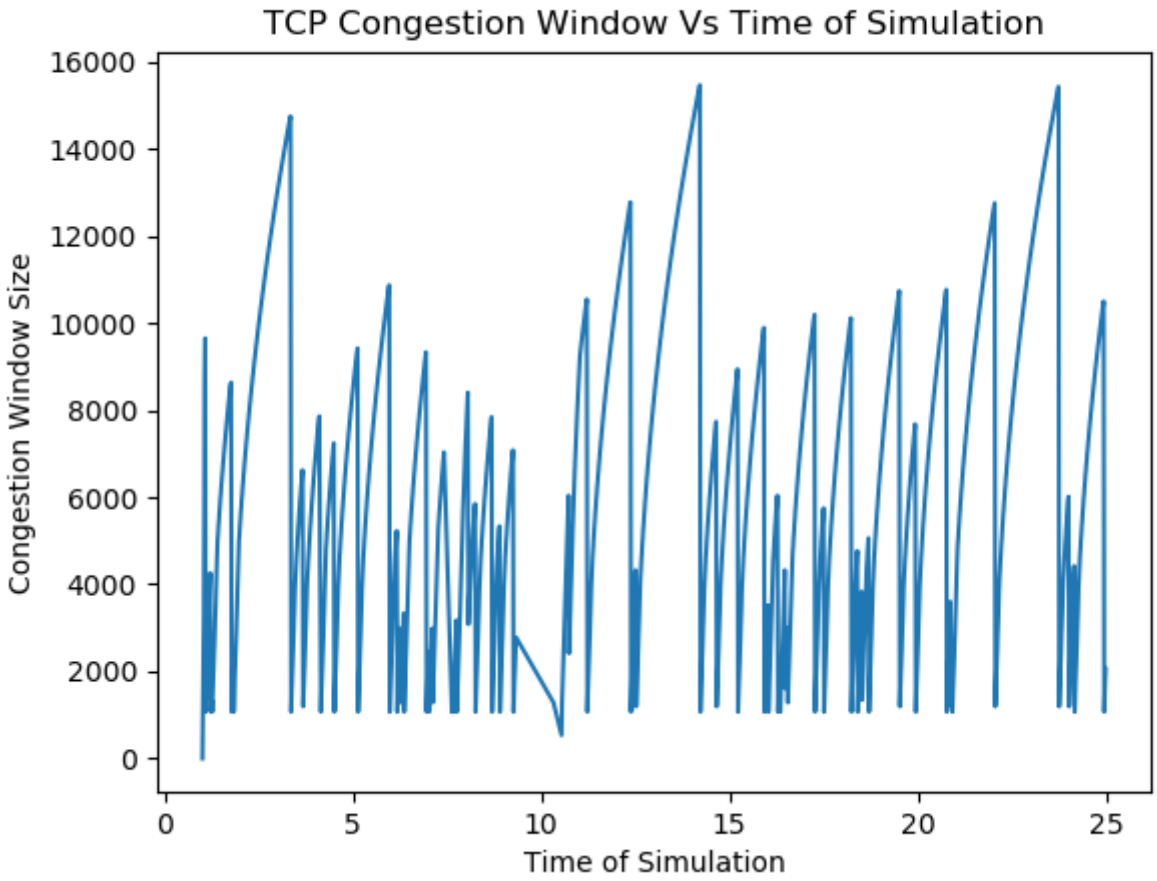
Description: This Tcp NewReno protocol is an extension of the Tcp Reno protocol and here the problem of Tcp Reno to do reduction of congestion window multiple times with each packet loss even though there was no need to do reduction was handled elegantly and the trick was to let the sender know that it needs to reduce the congestion window by 50% for all the packets loss that happened in the same congestion window.

Tcp Vegas:



Description: Tcp Vegas is a completely different approach to bandwidth management and it tries to control the congestion window before packet loss occurs by keeping track of the base RTT and the current RTT of the packets and the formula is :
$\alpha <= (expectedOutput - actualOutput) * baseRTT <= \beta$
where $expectedOutput =$ window size divided by baseRTT, $actualOutput =$ window size divided by currentRTT and baseRTT is the minimum RTT measured so far.
Here Tcp Vegas tries to keep the queue size in between α and β and whenever congestion window violates this range, it is accordingly incremented or decremented by 1 and in case of packet drop it is reduced to half.

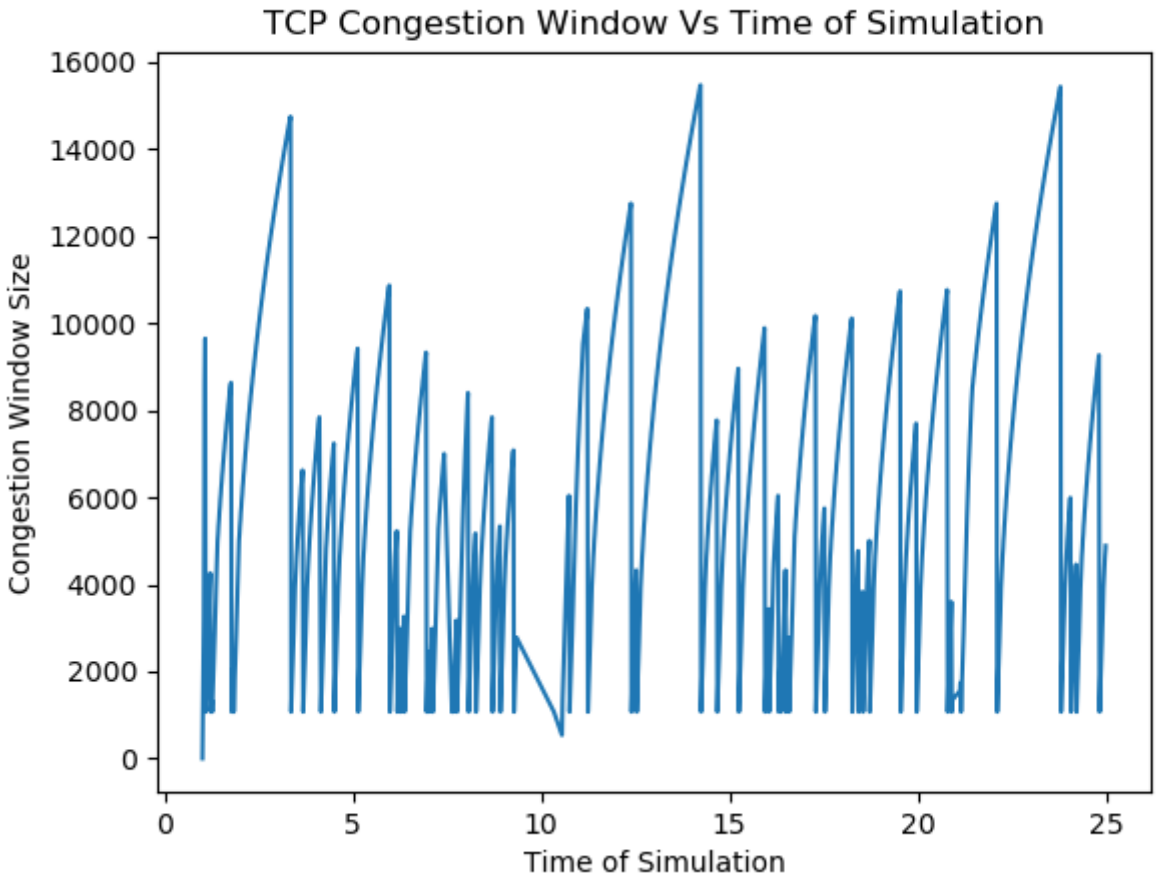
Tcp Veno:



Description: Tcp Veno protocol is an enhancement for transmission over wireless networks and it is like a combination of Tcp NewReno and Tcp Vegas since it computes the queue size as $N_{queue} =$ congestion*window - $bandwidth * baseRTT$ and the mechanism to change the congestion window size is :*
In the avoidance phase when there is no packet drop:
*if $N*{queue} < \beta$ then congestion*window = congestion_window + 1 each RTT*
*if $N*{queue}>=\beta$ then congestion*window = congestion_window + 0.5 each RTT*
and here β is the queue utilization value at which this Veno protocol slows down and this makes it different from Vegas since here even when in congestion avoidance phase, when $N*{queue} >= \beta$ then also congestion window size keeps on increasing unlike Vegas.

In the packet loss phase the changes on congestion window are:
if $N_{queue} < \beta$ then congestion*window = (4/5) * congestion_window*
*if $N*{queue} >= \beta$ then congestion_window = (1/2) * congestion_window*

Tcp Westwood:



Description: Tcp westwood is a sender side modification of Tcp NewReno protocol in order to handle the packet loss more effieciently by computing the congestion parameters like threshold size and congestion window size in a different manner.

3. Comparison of Graphs:

As we have already described the working of all the above protocols, it is now easy to see why we are getting these natures of the above graphs. Since this topology is simple in terms of node connections i.e. there are no intermediate routers to produce queuing, hence Tcp NewReno, Tcp Veno and Tcp Westwood show almost the same behaviour but in case of Tcp Vegas since it tries to maintain the size of congestion window between upper and lower bounds (calculated using baseRTT and bandwidth) in its congestion avoidance phase due to which we see a lower peak window size in case of Tcp Vegas.

4. BBR protocol:

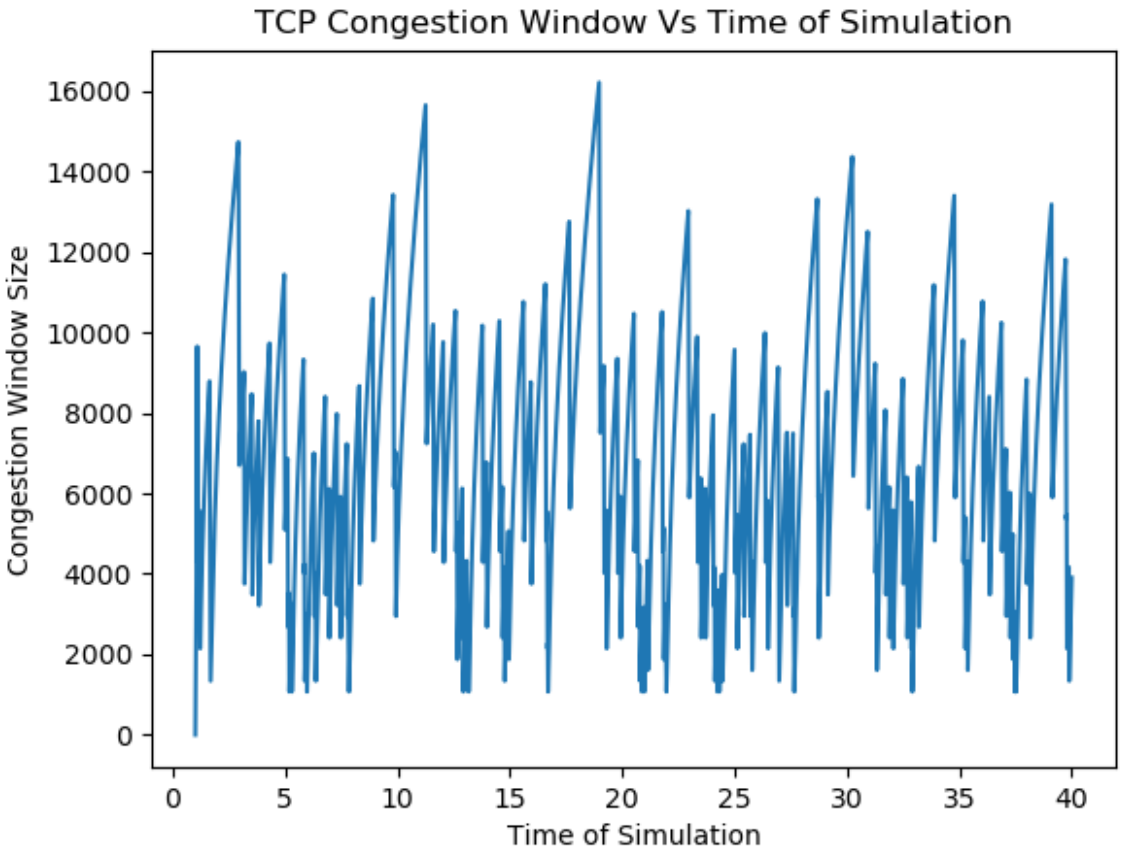
BBR introduced a new approach of congestion window size handling since even with minute packet losses, the above defined Tcp protocols may become unstable and in order to do so BBR uses latency instead of packet loss as parameter to adjust the congestion window size. It also avoids filling up the bottle neck buffer which can lead to buffering and high latency thus the name comes **BBR** = **B**ottleneck **B**andwidth and **R**oundtrip propogation time.

Task 2:

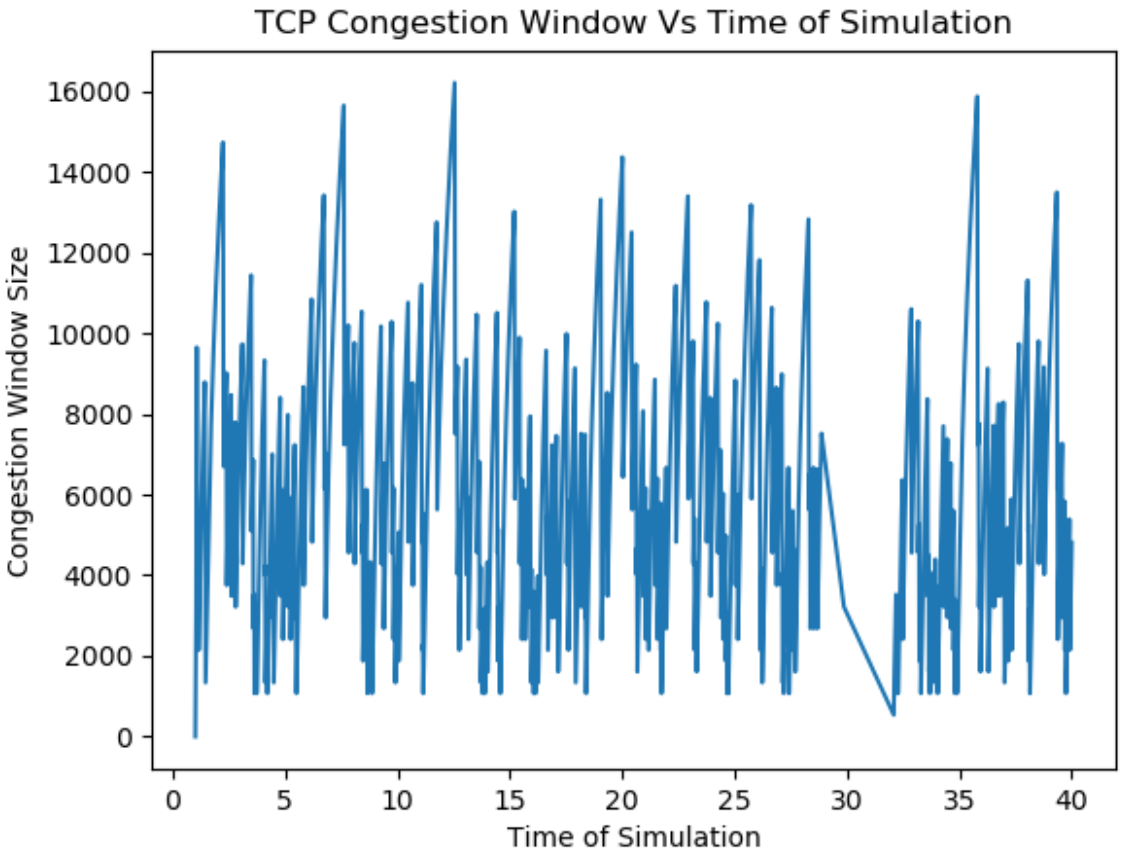
In this task we are gonna analyse the effect of application data rate and channel data rate on the congestion window size of Tcp source N1 with Tcp NewReno as the Tcp protocol.
Here the code we will use is present in the **task2.cc** file and this code is similar to the code used in Task 1 since here also we generate **task2.txt** file which is the read by the plotting script **task2_plot.py** which just reads the **task2.txt** file and creates the plot of congestion window with time.

(a) Application Data Rate : 5Mbps

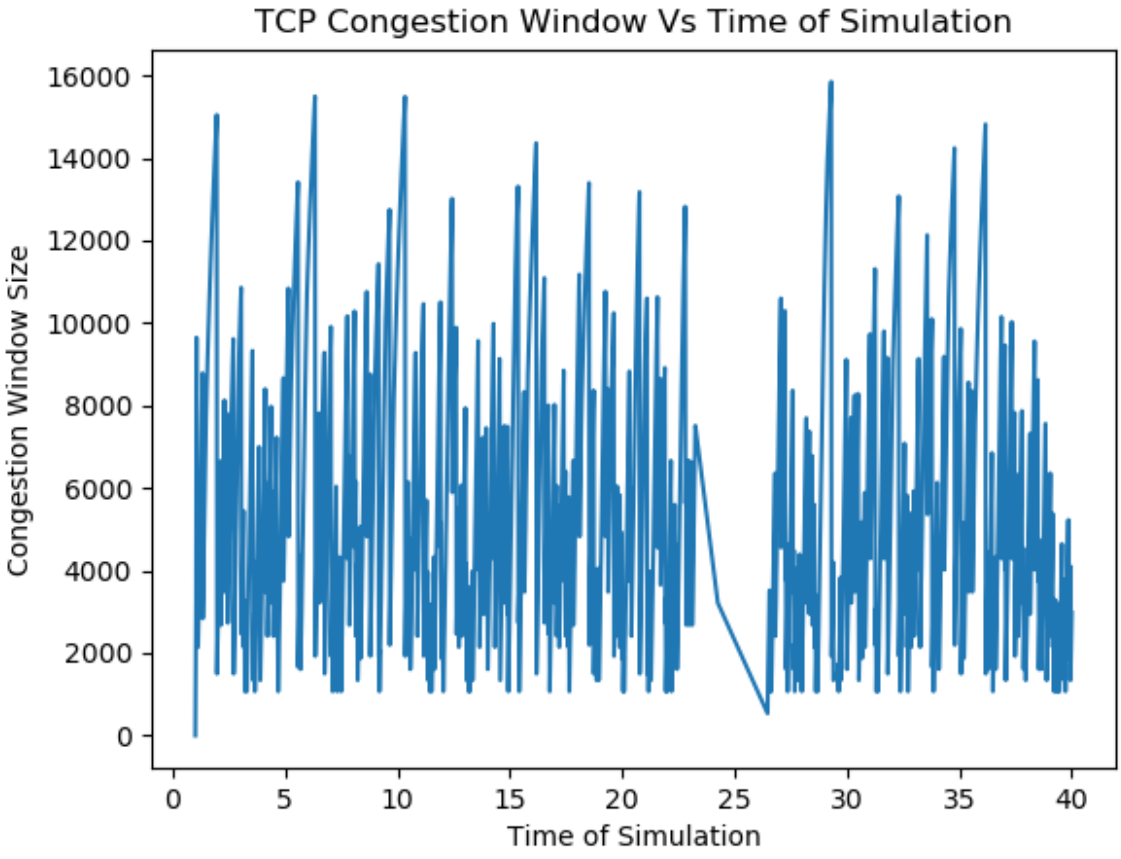
1. **Channel Data Rate : 3Mbps**
   Here the max window size is 16206 and the graph looks like:



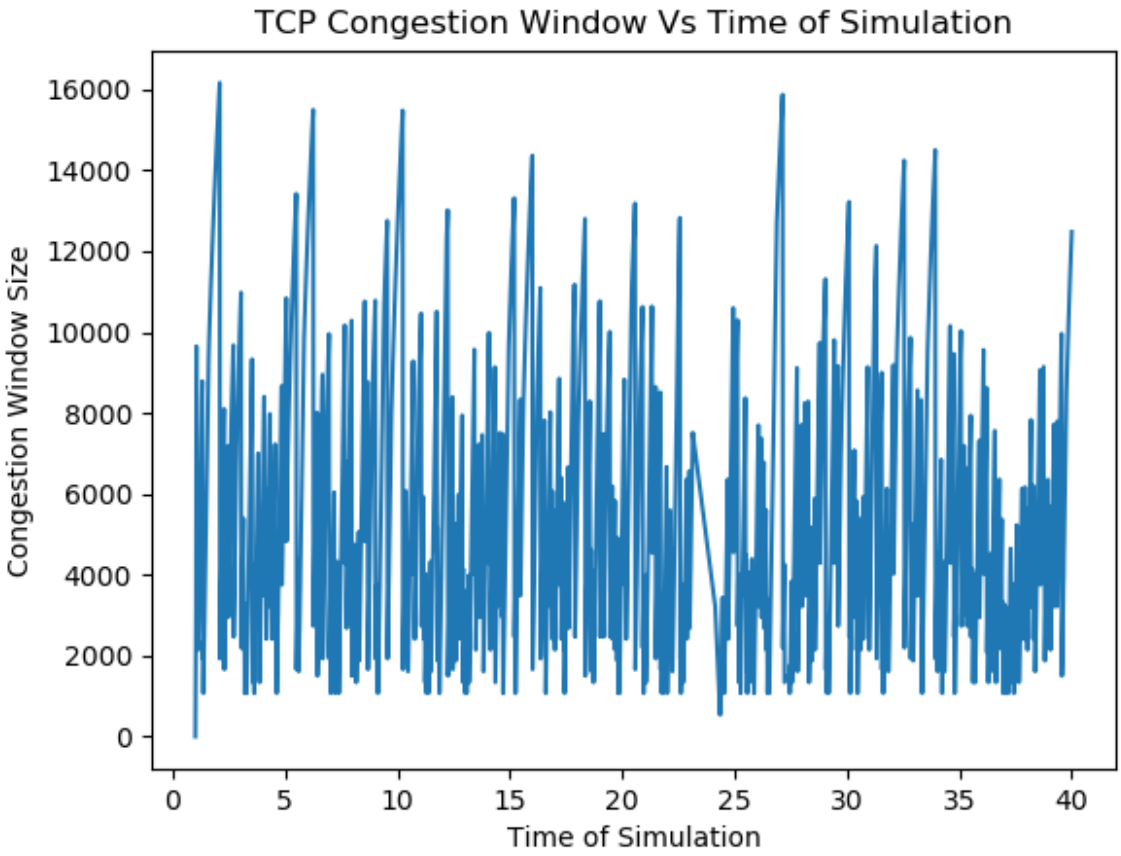
2. **Channel Data Rate : 5Mbps**
   Here the max window size is 16206 and the graph looks like:



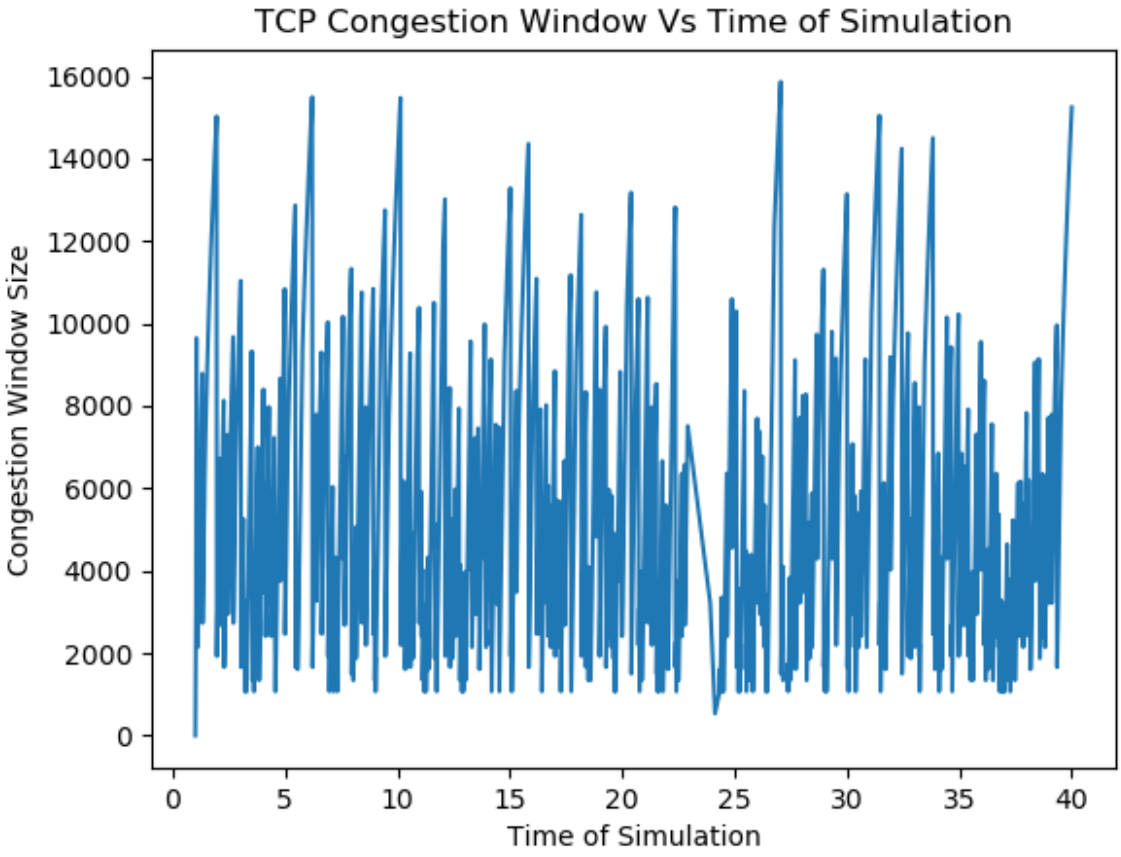
3. **Channel Data Rate : 10Mbps**
   Here the max window size is 15849 and the graph looks like:



4. **Channel Data Rate : 15Mbps**
   Here the max window size is 16162 and the graph looks like:

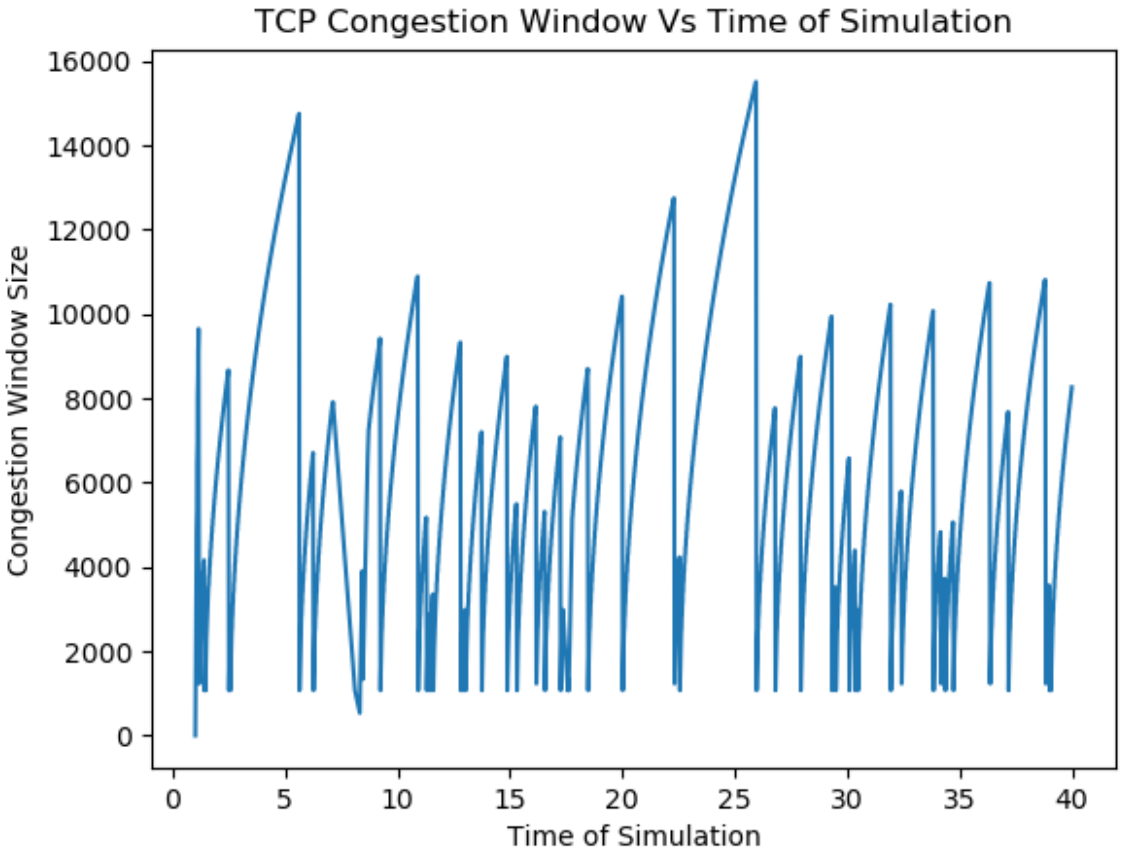


5. **Channel Data Rate : 30Mbps**
   Here the max window size is 15867 and the graph looks like:

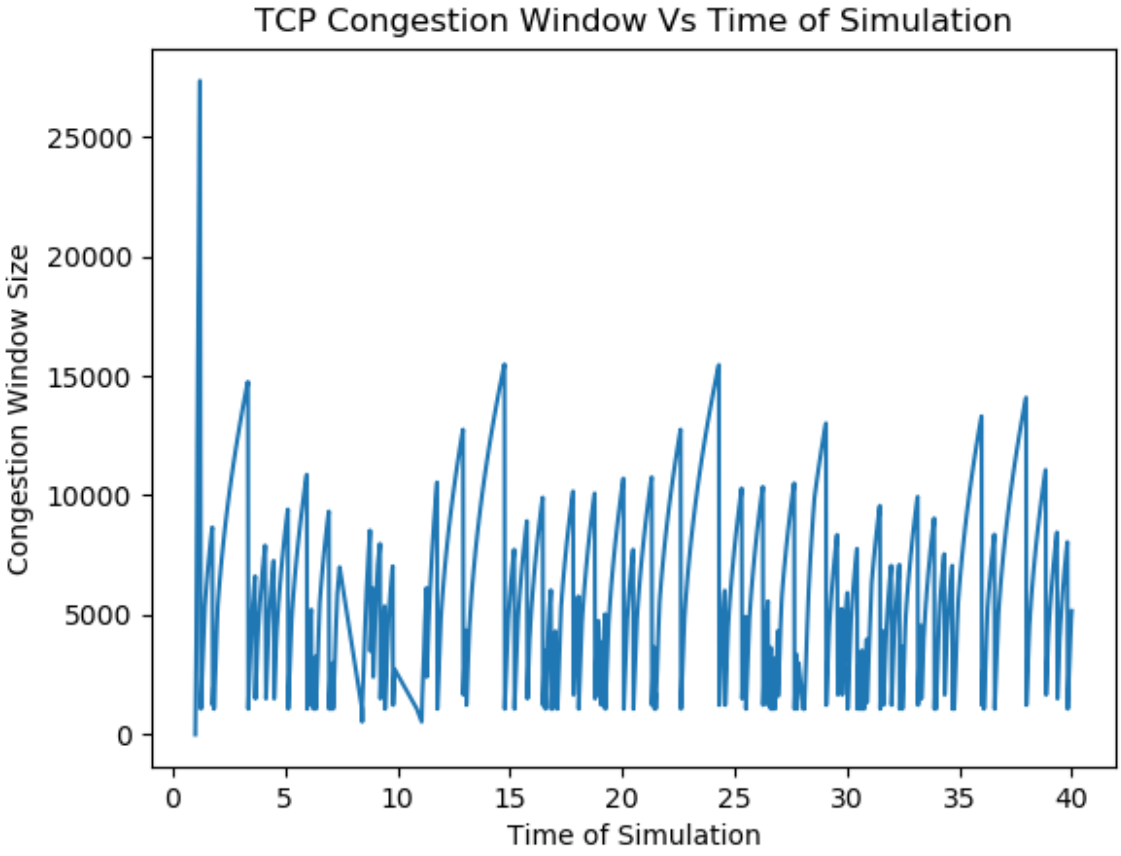


(b) Channel Data Rate : 4Mbps

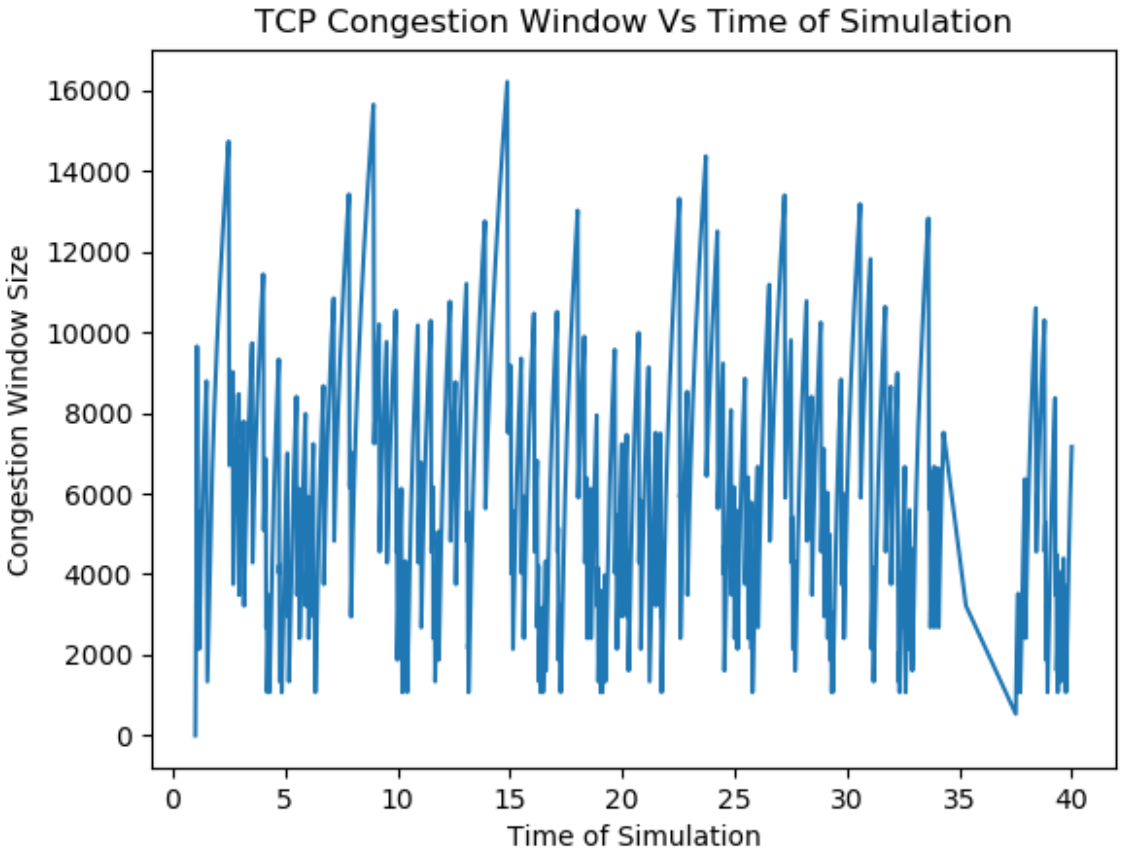
1. **Application Data Rate : 1Mbps**
   Here the max window size is 15507 and the graph looks like:



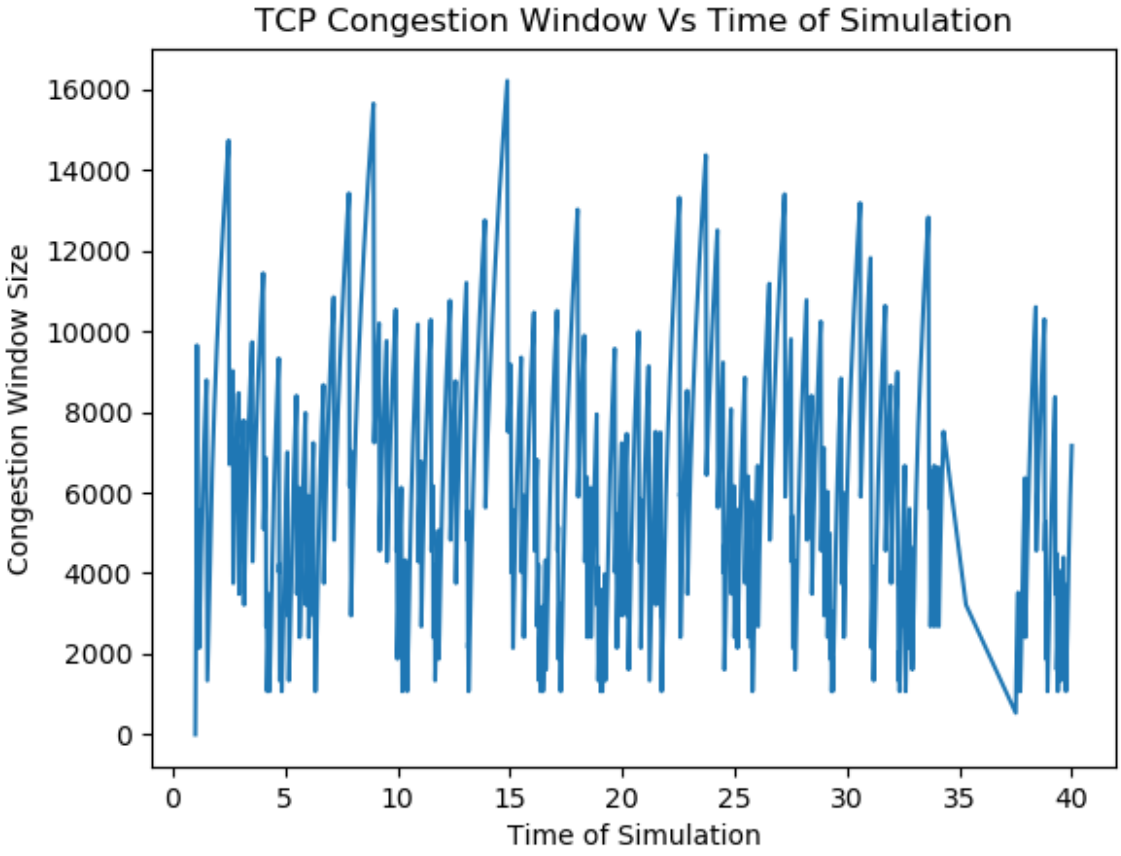
2. **Application Data Rate : 2Mbps**
   Here the max window size is 27336 and the graph looks like:



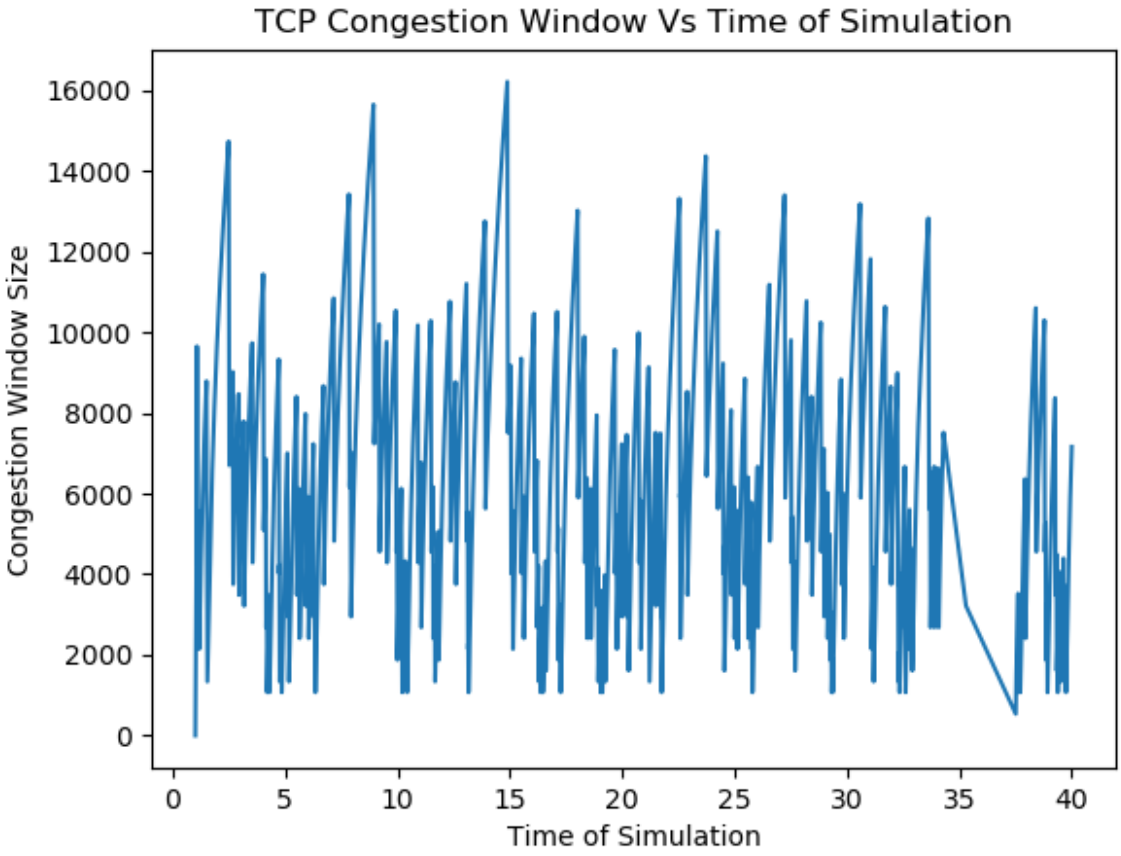
3. **Application Data Rate : 4Mbps**
   Here the max window size is 16206 and the graph looks like:



4. **Application Data Rate : 8Mbps**
   Here the max window size is 16206 and the graph looks like:



5. **Application Data Rate : 12Mbps**
   Here the max window size is 16206 and the graph looks like:



Observations:

Here our observation upon observing the above graphs is:

1. On increasing the application data rate, we see more number of packet drops and most of the time is utilised in resizing the congestion window due to these packet drops.
2. On increasing the channel data rate, we see that the congestion window size is increased when we are going from channel data rate 3Mbps to 5Mbps and beyond that this change is slightly observable since the application data rate is now the new bottleneck for data transfer.
3. The relationship we observe between channel data rate and application data rate is that when we have high application data rate and low channel data rate, then we see a significant amount of packet drop and when the application data rate is low and channel data rate is high then we don't have much impact on the congestion window size since the application data rate becomes the bottleneck in this case.

Task3:

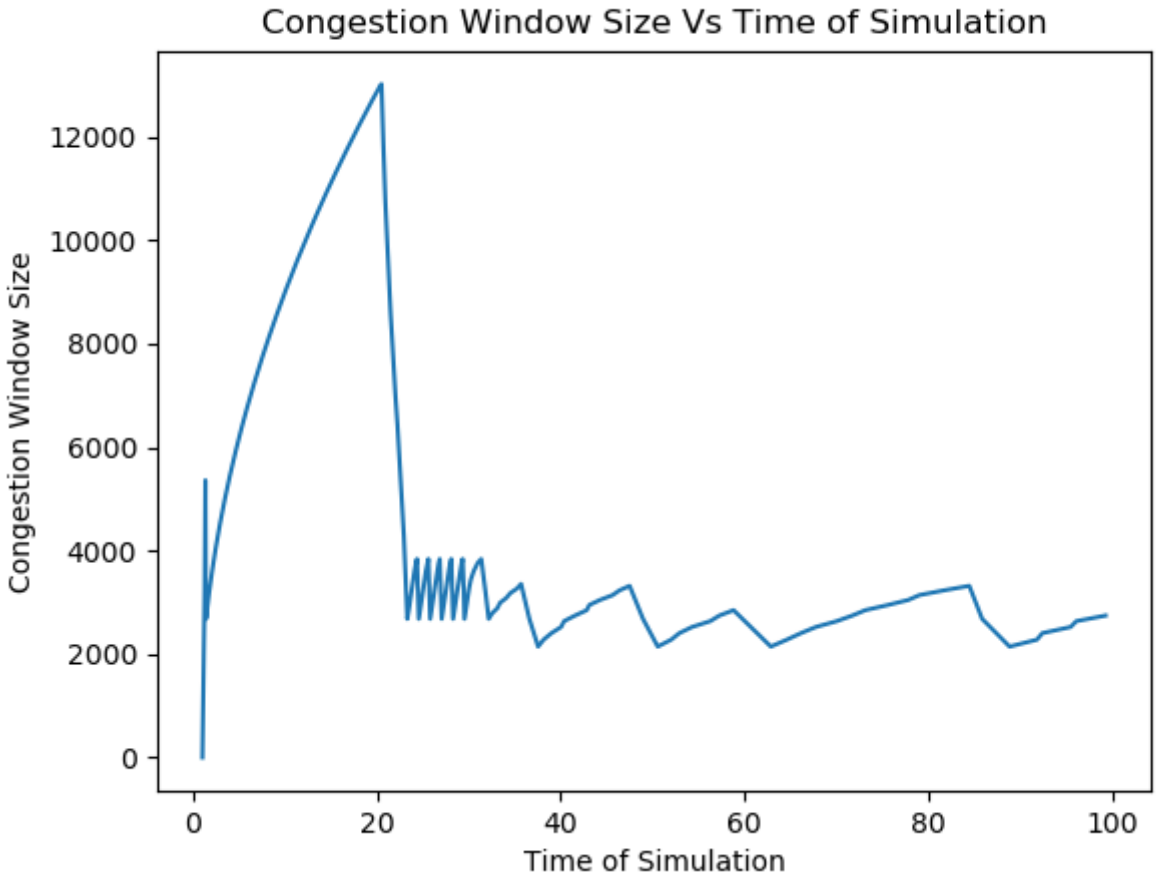
Here in this task, we will see the impact of UDP clogging at the intermediate router node on the size of Tcp congestion window sent from a source to a sink having this clogged router in its packet transfer path.

Code Explanation:

The code for this task can be found in the **task3.cc** file and upon running which, we will get a **task3.txt** file which contains the data for congestion window size at different time stamps upon reading which we will create the plot of Tcp congestion window vs Time of Simulation the script code of which is present in the **task3_plot.py** file. In order to create such a topology, I took the reference code from the seventh.cc example tutorial. The code explanation is as follows. First we set the default Tcp protocol to Tcp Vegas using the Config command. After which we will 5 nodes and install the internet stack on these nodes and we will create point to point links between different nodes, as described in the assignment diagram and we will store these point to point links in a vector so that we can access them at a later time and then we also store the corresponding net devices of these links in a similar fashion. Then we will do ip address assignment for these net devices and start Tcp application sink at node 4 and source at node 1. Simliarly we create Udp sink at node 5 and source at node 2 and run 2 applications on it (first one will run from 20 to 30 seconds and the second one will run from 30 seconds to 100 seconds with different application data rate to clog the intermediate Tcp packet transfer path router node). Finally we use pcap enabler to generate the pcap files corresponding to all the nodes.

(1) The plot of Congestion Window Vs Time for Tcp Simulation:

The graph for variation of Tcp congestion window vs time for simluation is as follows:



(2) Points of Observation for Effect in Congestion Window Size

The points where we see significant change in the above plot are first the point after some time of **20 seconds** where we can see that there is significant dip in the Tcp congestion window size then there is another point just ahead of **30 seconds** where we further see a slight dip in window size after a zig-zag pattern and here the first point occurs when the Udp packet transfer between node 2 and node 5 is started which clogs the node 2 and node 3 link resulting in packet loss for Tcp transfer from node 1 to node 4. Similarly the second point occurs when we further clog this link upto its capacity by increasing the application data rate due which further packet drops occur.

(3) Pcap Files

The pcap files are generated for all the net devices by using

```
pointToPoint.EnablePcapAll("task3");
```

this command.