CS 342: COMPUTER NETWORKS LAB

Assignment 4 - Network Simulation Using ns-3

GROUP NO: 52

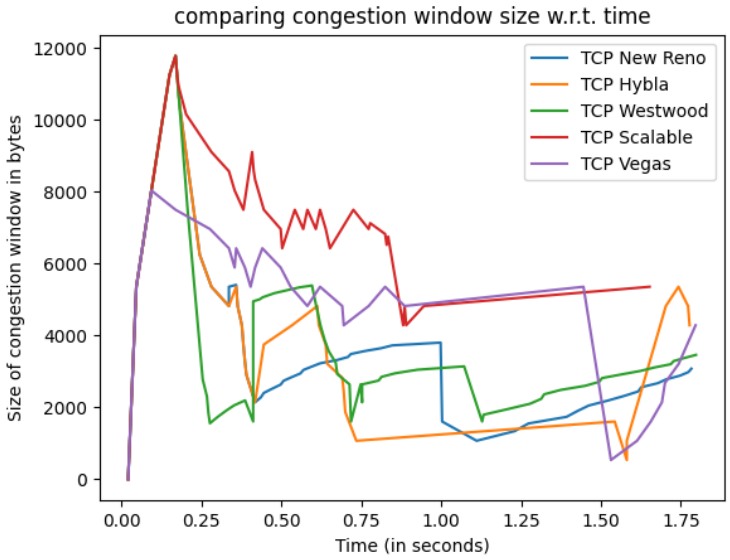
Sahil Kumar Gupta	200123081
Gaurang Thakur	200123082
Tiyasa Majumder	200123083

APPLICATION #3

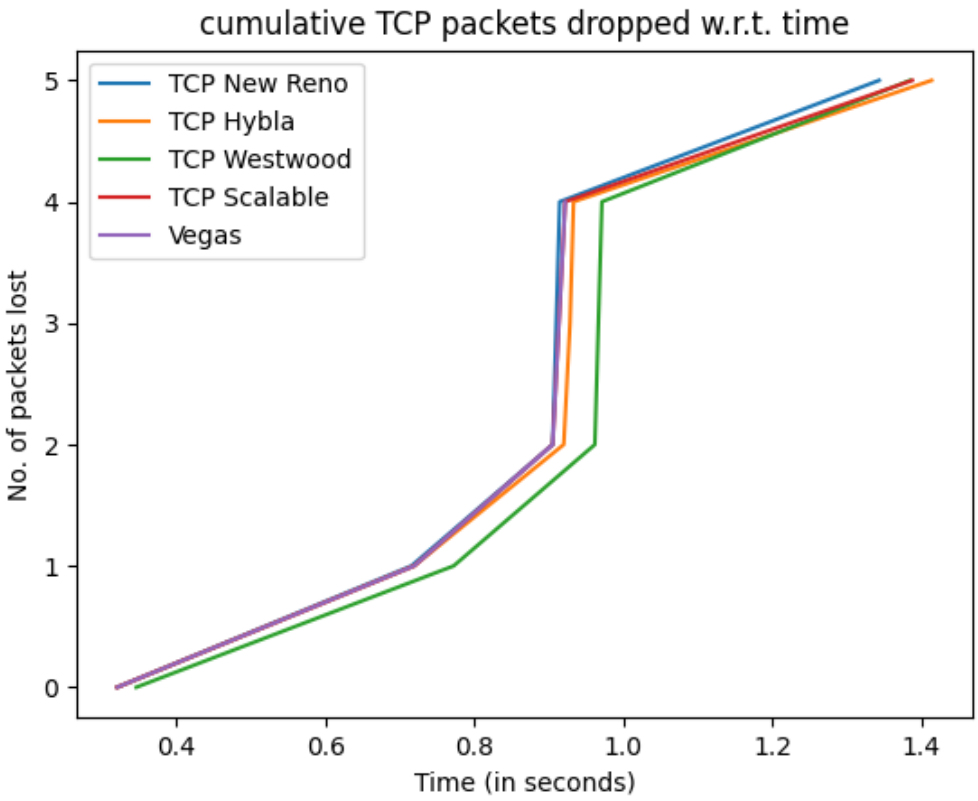
Comparison of congestion window size, packets dropped and bytes transferred w.r.t time for five different TCP Congestion Control Implementations

Note - The simulation has some unpredictability, thus the findings and charts shown in this report are not absolute and could vary from simulation to simulation.

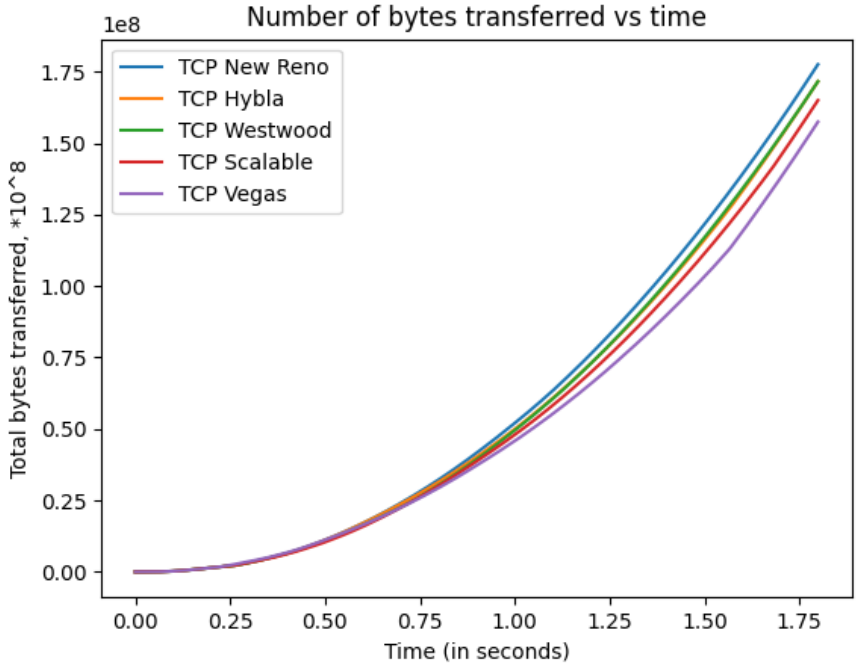
1) Graph of TCP congestion window w.r.t. time



2) Graph showing cumulative TCP packets dropped w.r.t. Time



3) Graph showing cumulative bytes transferred w.r.t. Time



As can be seen from the plot above, starting at 1, all TCP variations have an initial accelerated growth in the congestion window. Congestion/ loss is caused by the significant rise in congestion window size, which causes an increase in the number of packets being transmitted concurrently. The overlapping graphs in the initial section show that all the implementations perform nearly identically since they all use the same multiplicative increment policy, but when loss is discovered, they all act differently.

Observations:

Case 1: Using TCP New Reno

The window size will continue to increase until packet loss due to congestion occurs. If the window size was throttled because of packet loss, the connection's throughput would be reduced. This cannot be prevented because the congestion control method utilized in this specific TCP congestion control algorithm, TCP Reno, is important and can only identify network congestion through packet loss. However, when the TCP connection is the cause of the congestion because of its too large window size, throttling the window size is insufficient. If the window size is properly maintained, so that there is no packet loss in the network, throughput deterioration brought on by a throttled window can be avoided. As a result, TCP Vegas benefits more from the congestion window and of the network, and maintains a large congestion window size, which ultimately leads to increasing the throughput.

Case 2: Using TCP Hybla

Although TCP Hybla still follows the same peaks and throughputs as New Reno or Westwood, the congestion window peak there hits twice as soon. In many ways, it functions like TCP Reno. In order to avoid constantly entering and exiting the fast retransmit phase, NewReno modifies Reno-style protocols slightly by delaying exiting the phase until all acknowledgements of the outstanding sent data have been received. This might prevent the smooth and orderly construction of Westwood and New Reno.

Case 3: Using TCP Westwood

The plot shows that the congestion window with respect to time graphs of TCP Westwood and NewReno are pretty similar. This is observed from the plot and was also expected since TCP Westwood is a theoretical modification of NewReno. The window is managed by TCP Westwood using end-to-end rate estimation.

The main novel concept is to continuously estimate the packet rate of the connection at the TCP sender by observing the ACK reception rate.
After a congestion episode, the estimated connection rate is used to determine the congestion window and slow start threshold parameters.
It is superior to many traditional TCP implementations in that it analyses the bandwidth and adjusts the congestion window size accordingly, rather than just halving the congestion window size.

Case 4: Using TCP Scalable

TCP Scalable surpasses all other variants due to its design for high-speed networks with far faster recovery rates, as indicated by the plot's continuously huge congestion window sizes, which are observed in the red line of the graph. Scalable TCP causes changes in the congestion control mechanism. Up until the last packet loss, the congestion window size is reduced by a tiny fraction (a factor of 1/8 rather than normal TCP's 1/2) for each packet loss. When packet loss stops, the rate is ramped up at a sluggish fixed-rate (one packet is added for every 100 successful acknowledgments) rather than the Standard TCP rate, which is the inverse of the congestion window size. This causes huge windows to take a long time to recover.

When the round-trip time is 200 milliseconds, this helps cut the recovery time on 10 Gbit/s links from more than 4 hours (using Standard TCP) to less than 15 seconds.

The plot makes this point abundantly clear because Scalable TCP experiences much smaller drops in the congestion window (cwnd) than the other ones , and also the recovery is consistent.

Case 5: Using TCP Vegas

TCP Vegas, that is the purple line one , as is seen from the plot, maintains constant congestion window (cwnd) sizes. In contrast to other ones, TCP Vegas specifically does not accelerate its congestion window past a certain point. However, in contrast to the other variants, which exhibit sharp declines and high peaks, the variation maintains a fairly constant congestion window size. Contrary to other implementations like Reno, New Reno, etc., which only identifies congestion after it has really occurred via packet loss, TCP Vegas identifies congestion at an early stage based on growing Round-Trip Time (RTT) values of the packets in the connection.

INSTRUCTIONS TO RUN THE FILE

1. Place the lab_4.cc file in the scratch folder.

2. Open a terminal in the ns3 directory and run the corresponding command for which TCP type you want to run the program:

```
./ns3 run scratch/lab_4 -- --tcp_protocol_name=TcpNewReno
./ns3 run scratch/lab_4 -- --tcp_protocol_name=TcpHybla
./ns3 run scratch/lab_4 -- --tcp_protocol_name=TcpWestwood
./ns3 run scratch/lab_4 -- --tcp_protocol_name=TcpScalable
./ns3 run scratch/lab_4 -- --tcp_protocol_name=TcpVegas
```

If no tcp_protocol_name is specified, then TcpNewReno will run as it is the default tcp_protocol_name.

3. Files containing congestion window size data, dropped packets data and transferred bytes data will be created in the ns3 folder, for the TCP type for which the program has been run.

After that these values can be used to obtain the plot using the python code files submitted.