CS342 NETWORKS LABORATORY

ASSIGNMENT 4

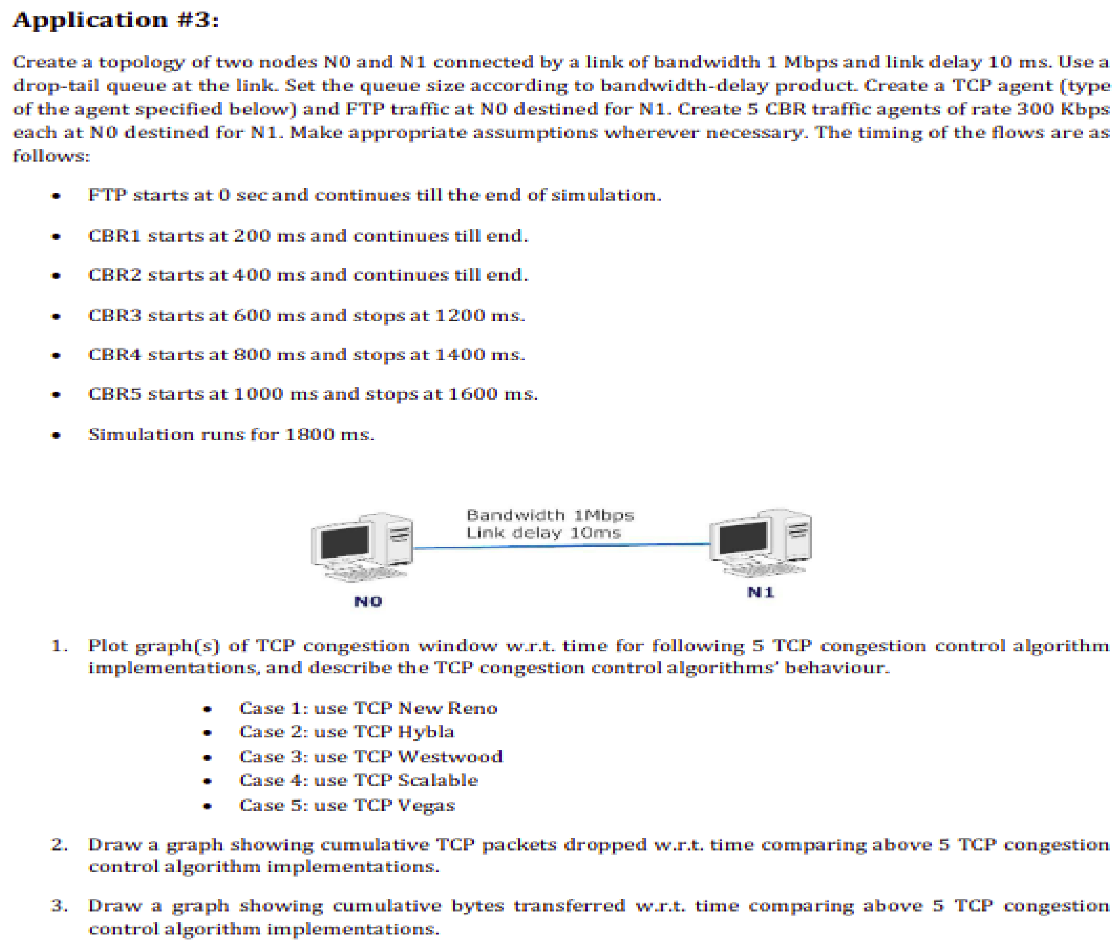
SUBMITTED BY:

GROUP 19

- SAMAY VARSHNEY, Roll no. 180101097
- BHARGAB GAUTOM, Roll no. 180123008

Note: There is a degree of randomness in the simulation, hence the results and the plots submitted in this report are not absolute, and may differ from simulation to simulation.

Question:



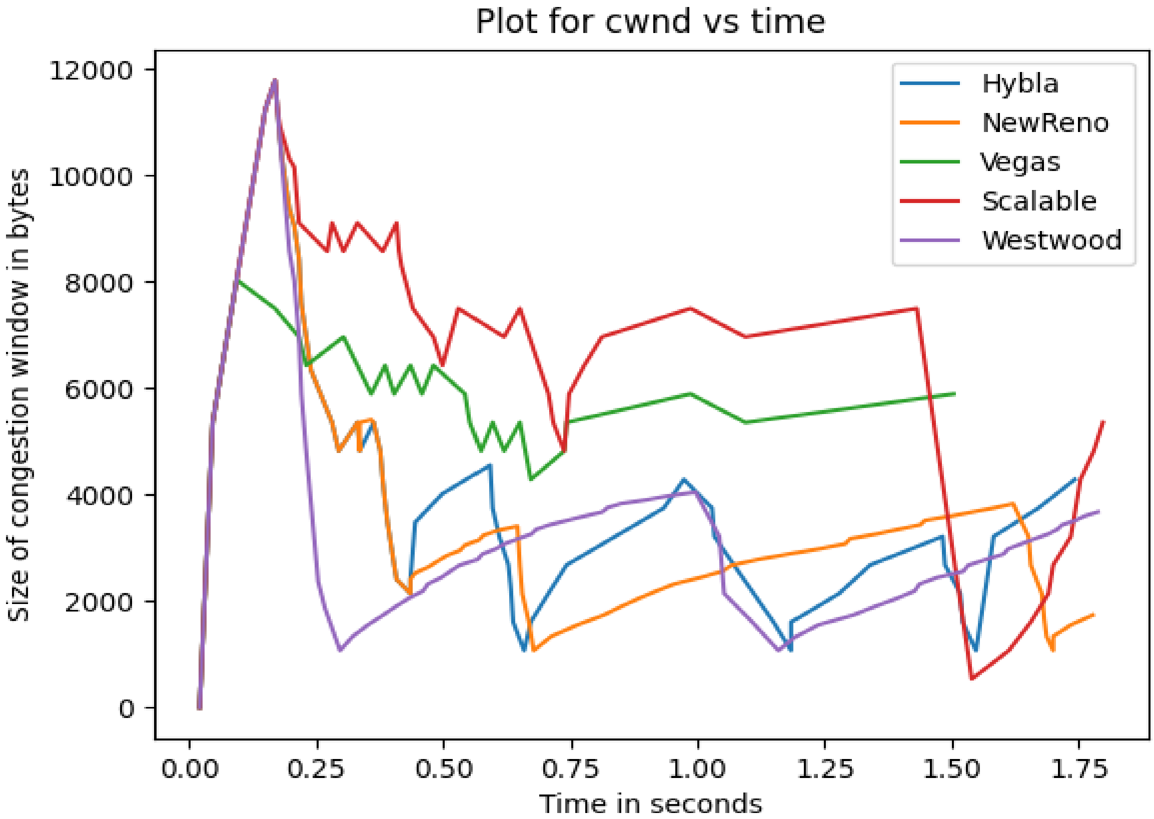
Application #3:

Create a topology of two nodes N0 and N1 connected by a link of bandwidth 1 Mbps and link delay 10 ms. Use a drop-tail queue at the link. Set the queue size according to bandwidth-delay product. Create a TCP agent (type of the agent specified below) and FTP traffic at N0 destined for N1. Create 5 CBR traffic agents of rate 300 Kbps each at N0 destined for N1. Make appropriate assumptions wherever necessary. The timing of the flows are as follows:

- FTP starts at 0 sec and continues till the end of simulation.
- CBR1 starts at 200 ms and continues till end.
- CBR2 starts at 400 ms and continues till end.
- CBR3 starts at 600 ms and stops at 1200 ms.
- CBR4 starts at 800 ms and stops at 1400 ms.
- CBR5 starts at 1000 ms and stops at 1600 ms.
- Simulation runs for 1800 ms.

1. Plot graph(s) of TCP congestion window w.r.t. time for following 5 TCP congestion control algorithm implementations, and describe the TCP congestion control algorithms' behaviour.
    - Case 1: use TCP New Reno
    - Case 2: use TCP Hybla
    - Case 3: use TCP Westwood
    - Case 4: use TCP Scalable
    - Case 5: use TCP Vegas
2. Draw a graph showing cumulative TCP packets dropped w.r.t. time comparing above 5 TCP congestion control algorithm implementations.
3. Draw a graph showing cumulative bytes transferred w.r.t. time comparing above 5 TCP congestion control algorithm implementations.

1) Graph of TCP Congestion Window vs Time



As its evident from the above plot, all the TCP variants enter into an initial accelerated increase in Congestion Window starting from 1. This heavy increase in Congestion window size results in an increasing number of packets being transmitted at a same time leading to congestion/loss. The Variants behave almost identically in the initial section (evident from the overlapping graphs in the initial part) as the multiplicative increment policy is same in all of them, however, they behave differently when loss is detected.

As, it is clear from the plot, TCP Vegas (green line) maintains consistent congestion window(cwnd) sizes. **TCP Vegas** particularly, does not accelerate its congestion window beyond a point unlike others. However, the variant maintains fairly consistent congestion window size unlike the others which show heavy drops and high peaks. TCP Vegas detects congestion at an incipient stage based on increasing Round-Trip Time (RTT) values of the packets in the connection unlike other flavors such as Reno, New, Reno, etc., which detect congestion only after it has actually happened via packet loss.

In **TCP NewReno** the window size continues to be increased until packet loss occurs due to congestion. When the window size is throttled because of packet loss, the throughput of the connection would be degraded. It cannot be avoided because of an essential nature of the congestion control mechanism adopted in TCP Reno; it can detect network congestion only by packet loss. However, throttling the window size is not adequate when the TCP connection itself causes the congestion because of its too large window size. If the window size is appropriately controlled such that the packet loss does not occur in the network, the throughput degradation due to the throttled window can be avoided. In this way, TCP Vegas takes greater benefit of the congestion window and of the network and maintains fairly large congestion window size effectively increasing the throughput.

TCP Scalable performs better than all of the other variants as its designed for high-speed networks with much faster recovery rates, this is quite evident from the high consistently high congestion window sizes in the plot.

Scalable TCP modifies the congestion control algorithm. Instead of halving the congestion window size, each packet loss decreases the congestion window by a small fraction (a factor of 1/8 instead of Standard TCP's 1/2) until packet loss stops. When packet loss stops, the rate is ramped up at a slow fixed rate (one packet is added for every one hundred successful acknowledgements) instead of the Standard TCP rate that's the inverse of the congestion window size (thus very large windows take a long time to recover). This helps reduce the recovery time on 10 Gbit/s links from 4+ hours (using Standard TCP) to less than 15 seconds when the round-trip time is 200 milliseconds.

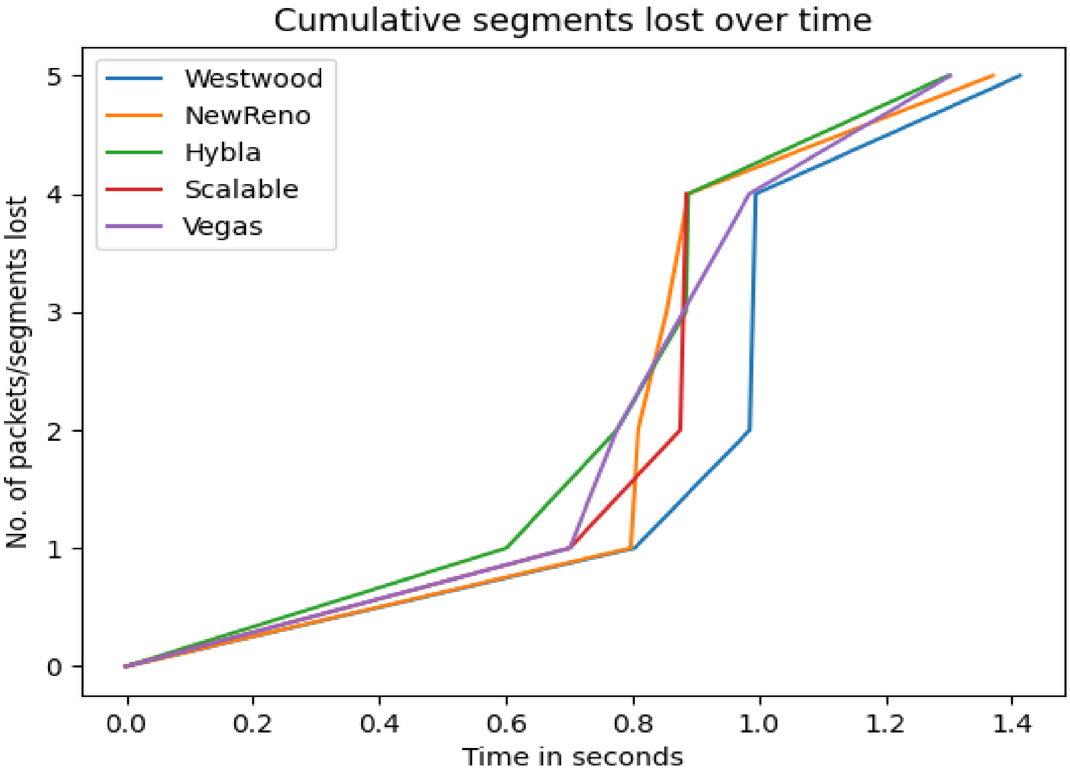
This is clearly evident from the plot as the drops in congestion window(cwnd) in case of Scalable are much smaller than the other variants and the recovery is consistent every time.

TCP Westwood and **NewReno** are quite similar in terms of their congestion window stats as is observed from the plot. Theoretically, TCP Westwood is a modification of NewReno and hence this observation was expected.

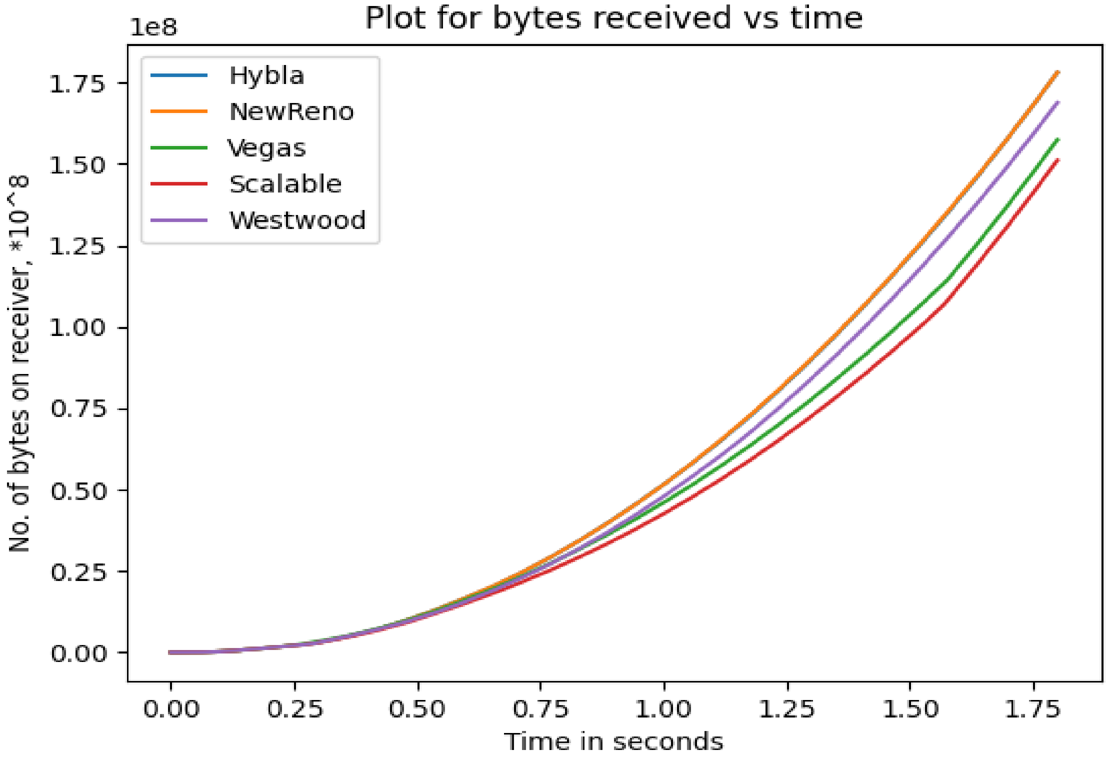
TCP Hybla also follows the same peaks and throughs as the other two, however the congestion window peak reaches twice as fast as either in NewReno or Westwood. Typically, it behaves like TCP Reno.

NewReno actually is a slight modification of Reno style protocols in that it that it does not escape the fast retransmit process till all the acknowledgements of the outstanding sent data is received, this prevents it from jumping in an out of the phase multiple times. This may be a possible reason for the Westwood and NewReno building up smoothly.

2) Graph of Cumulative TCP Packet Loss vs Time



3) Graph of Cumulative Bytes Transferred/Received vs Time



****