CS349: Assignment 4

Application 3

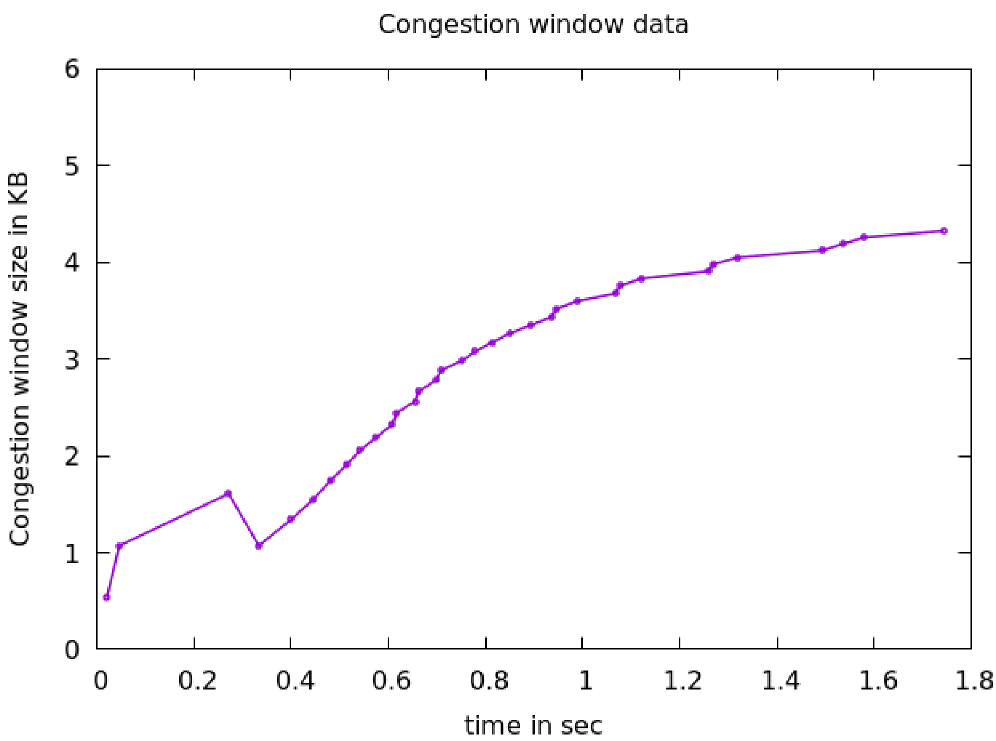
Group – 15
Soumik Paul – 170101066
Aayush Patni – 170101001
Rashi Singh - 170101052

Network Topology

node 0 ----------- node 1
1 Mbps
10 ms

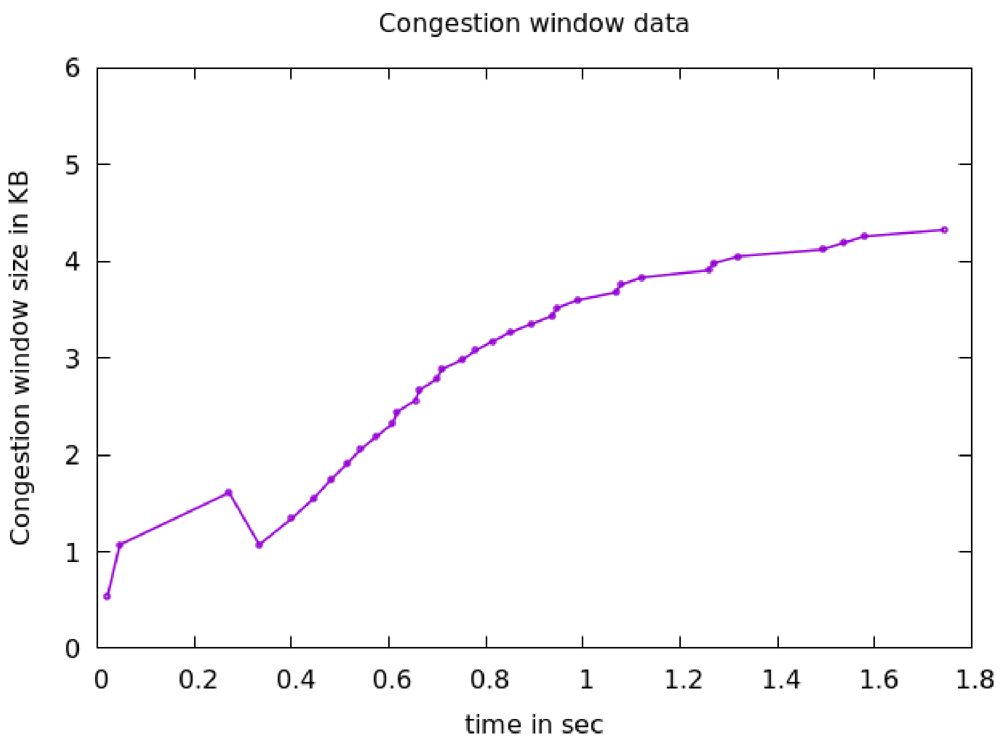
Congestion window vs Time Graphs

1. **TCP New Reno**



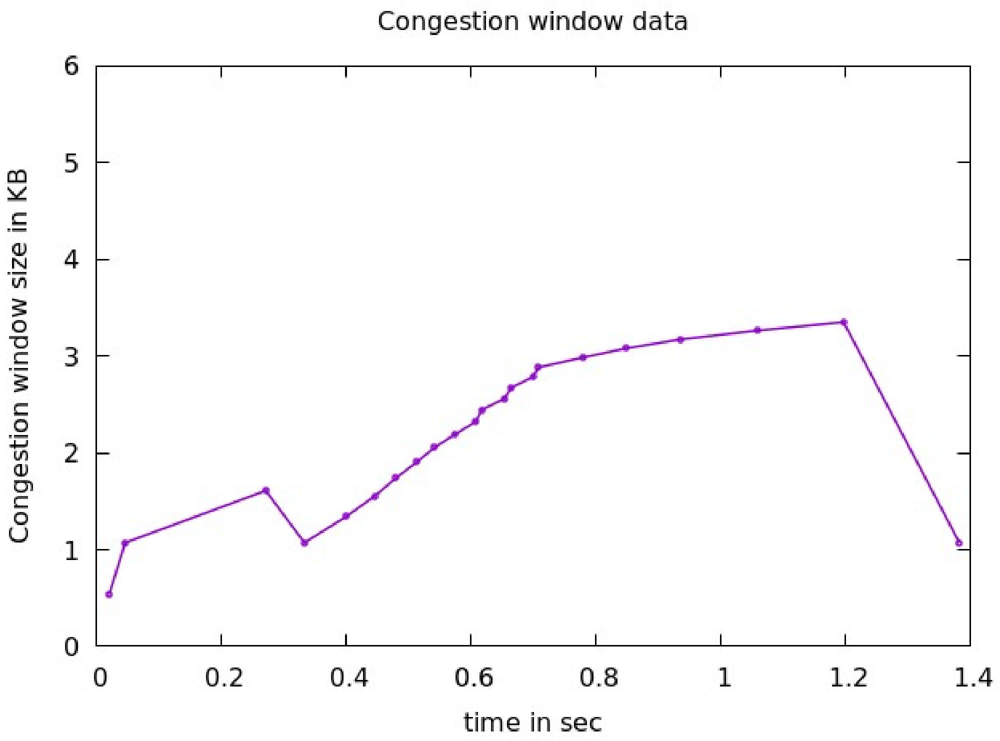
- Slow start between 0-0.2 seconds, where cwnd grows exponentially per RTT. Congestion avoidance from before 0.2 sec and untill about 0.3 sec where cwnd increases linearly. Enters fast recovery state at about 0.3 sec and cwnd decreased to about half it's size. After successful fast transmission, it enters congestion avoidance state and increases linearly with each ACK.
- Reason - During fast recovery, to keep the transmit window full, for every duplicate ACK that is returned, a new unsent packet from the end of the congestion window is sent. For every ACK that makes partial progress in the sequence space, the sender assumes that the ACK points to all the following packets as lost, and the next packet beyond the ACKed sequence number is sent. This allows TCP New Reno to overcome mutiple losses without waiting for a retransmission timeout or re-enter fast retransmit.

2. **TCP Westwood**



- Observed graph is very similar to NewReno (as explained above) for the given simulation.
- Reason – Westwood is a sender-side-only modification of New Reno, intended for links with large bandwidth-delay product. It relies on *mining* ACK stream to set values to parameters like slow start threshold and congestion window. What this means is that Westood keeps track of ACK reception andd used it to estimate bandwidth and "Eligible data rate" for the link. The resultant performance gains in efficiency, without undue sacrifice of fairness, also known as "opportunistic friendliness".

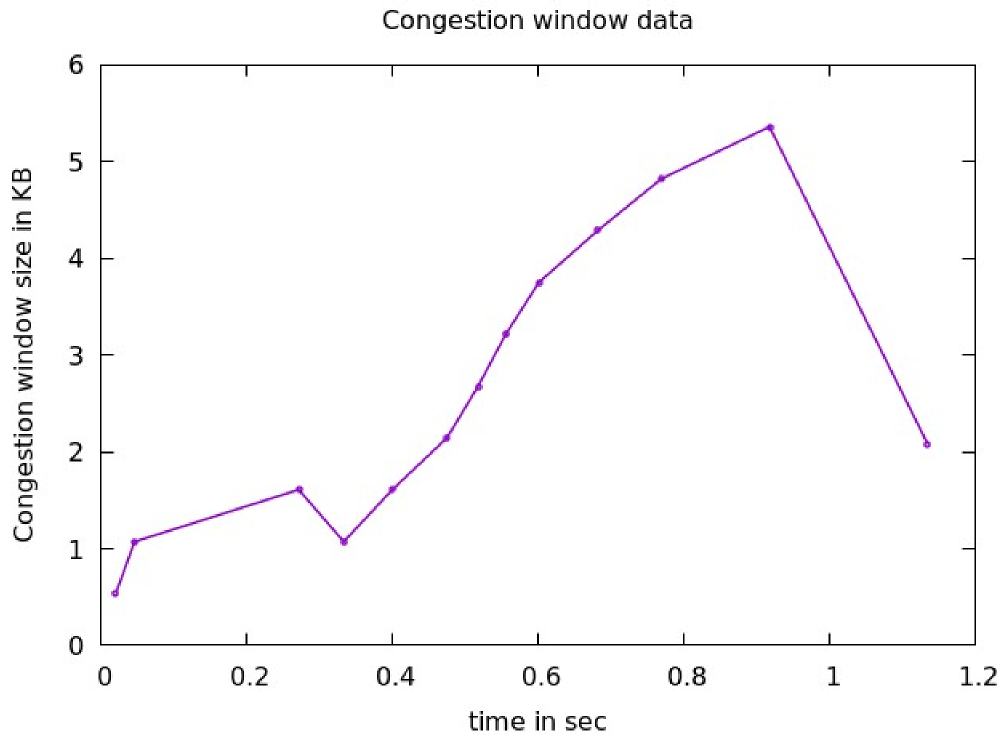
3. **TCP Vegas**



- Similar to before, there is slow start at the begining, followed by congestion avoidance and fast recovery. After 0.4 sec, there is a (roughly) linear increase in window size (congestion avoidance state). At around 1.2 sec, we se a sharp decease in cwnd as it again enters fast recovery (unline New Reno).
- Reason – Unline NewReno, that detect congestion *after* packet loss occurs, TCP Vegas uses RTT values to detect congestion at an incipient stage. Performace of Vegas degrades in comparision to Reno/New Reno as it detects congestion early and reduces sending rate before Reno. The detect onset of congestion , Vegas compares Expected throughput (= window size/RTT) and actual throughput (= ACKs/RTT).
    - If actual < expected < actual + α
        - decrease cwnd to increase throughput
    - if actual + α < expected < actual + β
        - do nothing
    - if expected > actual + β
        - increase cwnd to decrease data rate before packet drop.

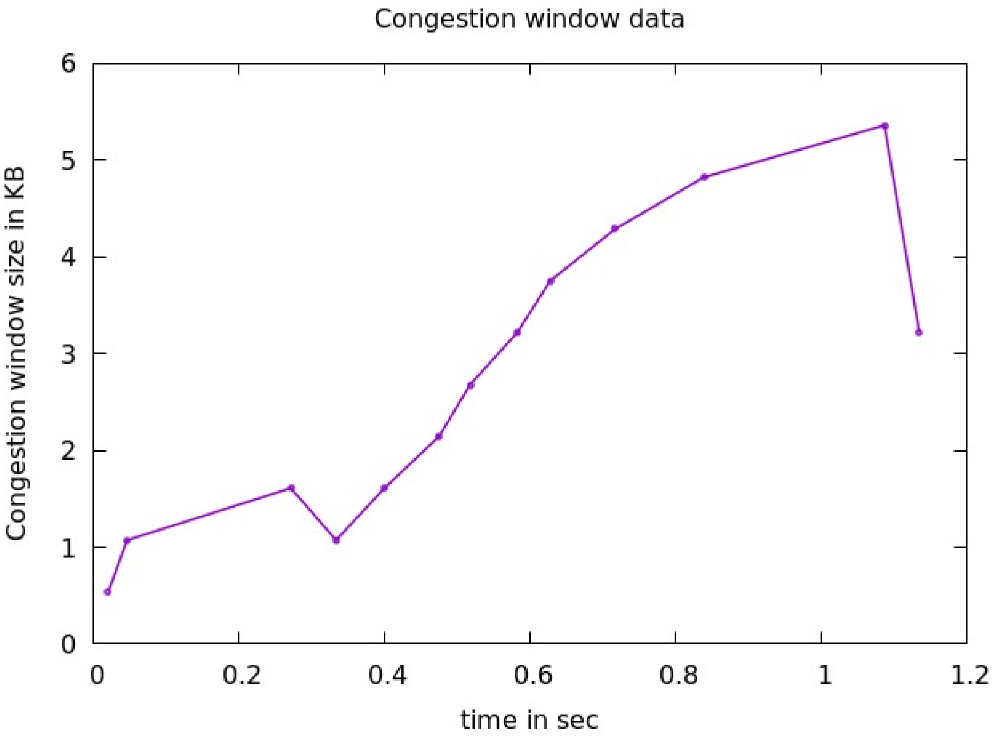
  (Thresholds of α and β correspond to how many packets Vegas is willing to have in window)

4. **TCP Hybla**



- Similar to Vegas, there is a slow start. Congestion avoidance and fast recovery stage before 0.4 sec. After that we see somewhat non-linear increase in cwnd and then a shapr decrease at the end.
- Reason – In TCP New Reno, links with more RTT stay in Slow Start state for a longer time, resulting in a “cwnd advantage” to those with smaller RTT that exit SS state earlier and achieve greater cwnd in avoidance (linear increase) state. To tackle this, TCP Hybla uses a cwnd growth formula that increases congestion window side independent of RTT. This results in all TCP connections to *exit* slow start phase at the *same time* (approximately). In avoidance state, there is more increase incase of larger RTT, to compensate cwnd advantage. Therefore a throughput (almost) independent of RTT is achieved.

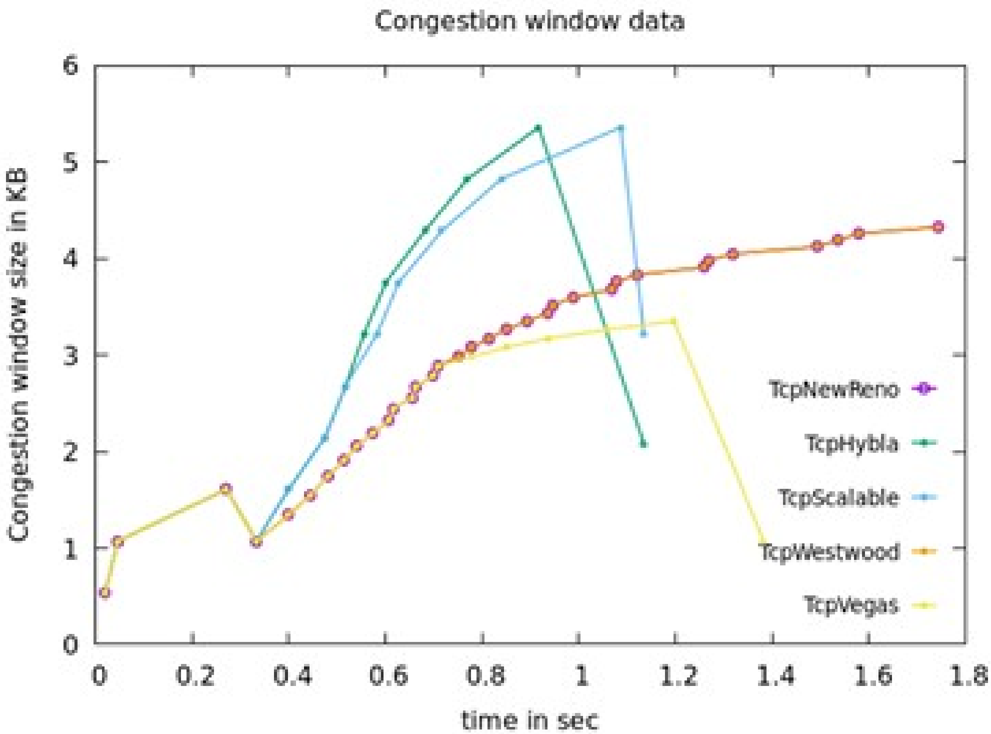
5. **TCP Scalable**



- Similar trend as Hybla for this simulation, but not a very steep decrease towards the end.
- Reason – TCP Scalable looks to increase throughput by modifying the congestion control algorithm as follows. Instead of halving the congestion window size, each packet loss decreases the congestion window by a small fraction (a factor of 1/8) 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 results in a fixed increase in cwnd independent of RTT.

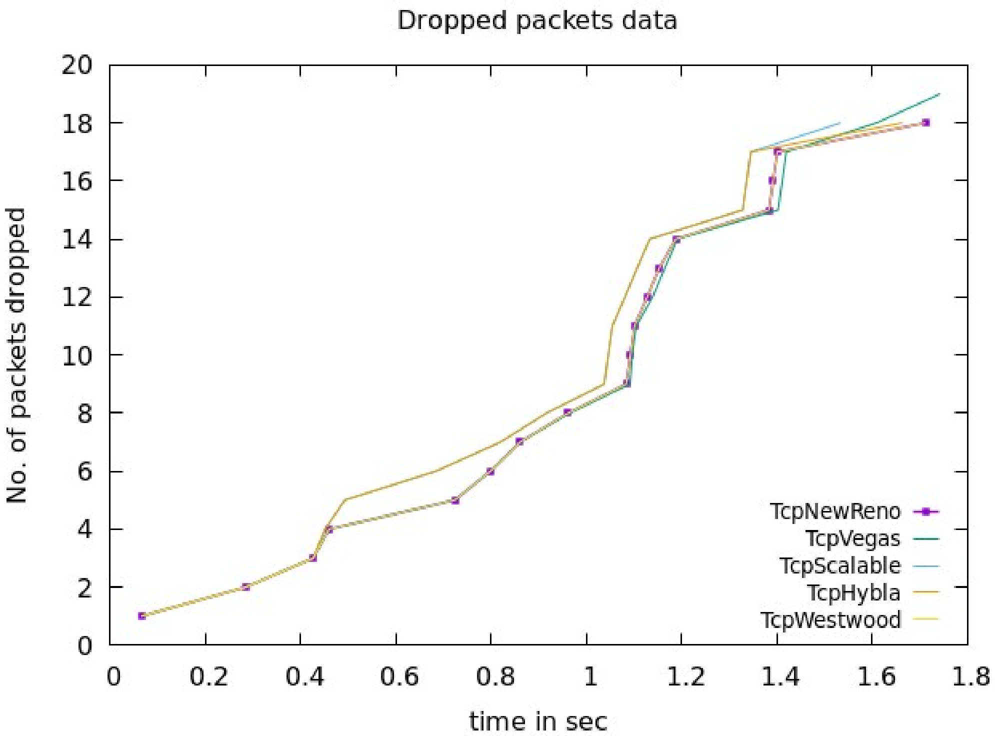
CONCLUSION

We observe similar trends based on whether cwnd size is changed using packet loss, or RTT (delay).**TCP Scalable**



Cumulative Packets Dropped vs Time

TCP Scalable



Cumulative Bytes transferred vs Time