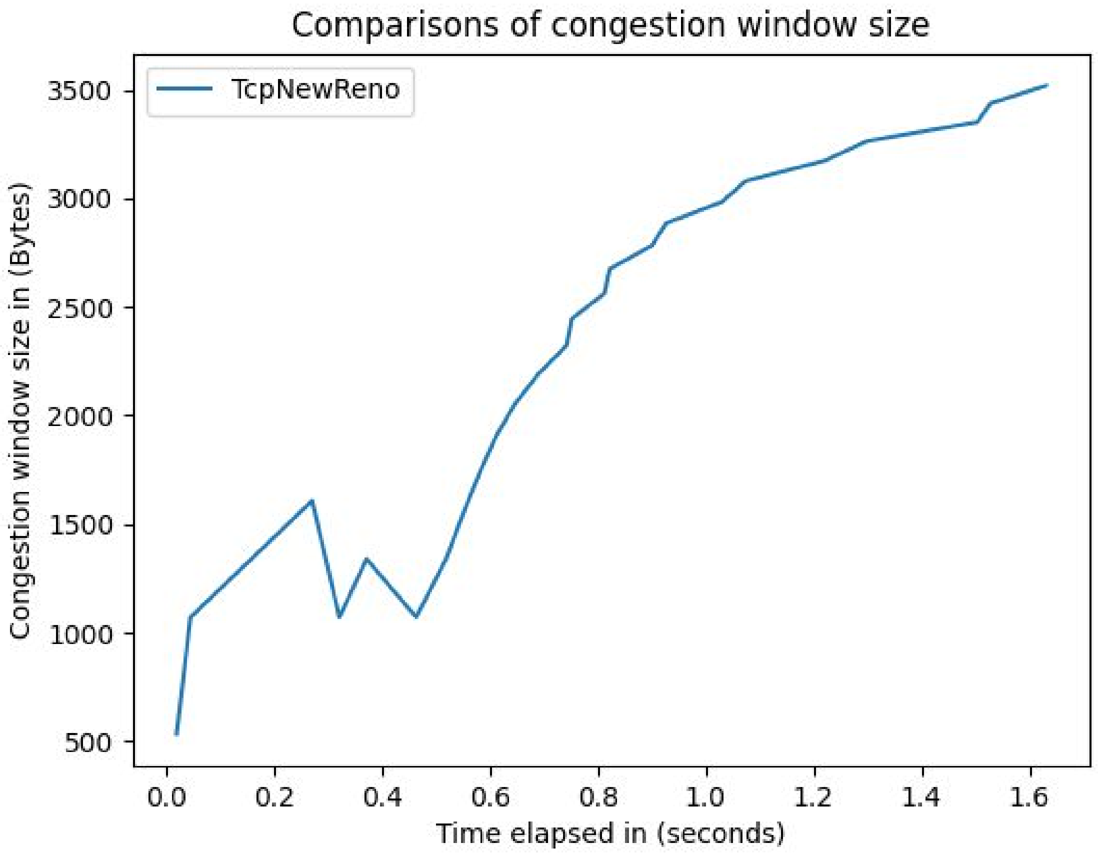
CS 341 Assignment 4
Dristiron Saikia(180101022), Nikunj Heda(180101049)
Group No. 20

<u>Congestion window v/s Time graph for TCP variants</u>

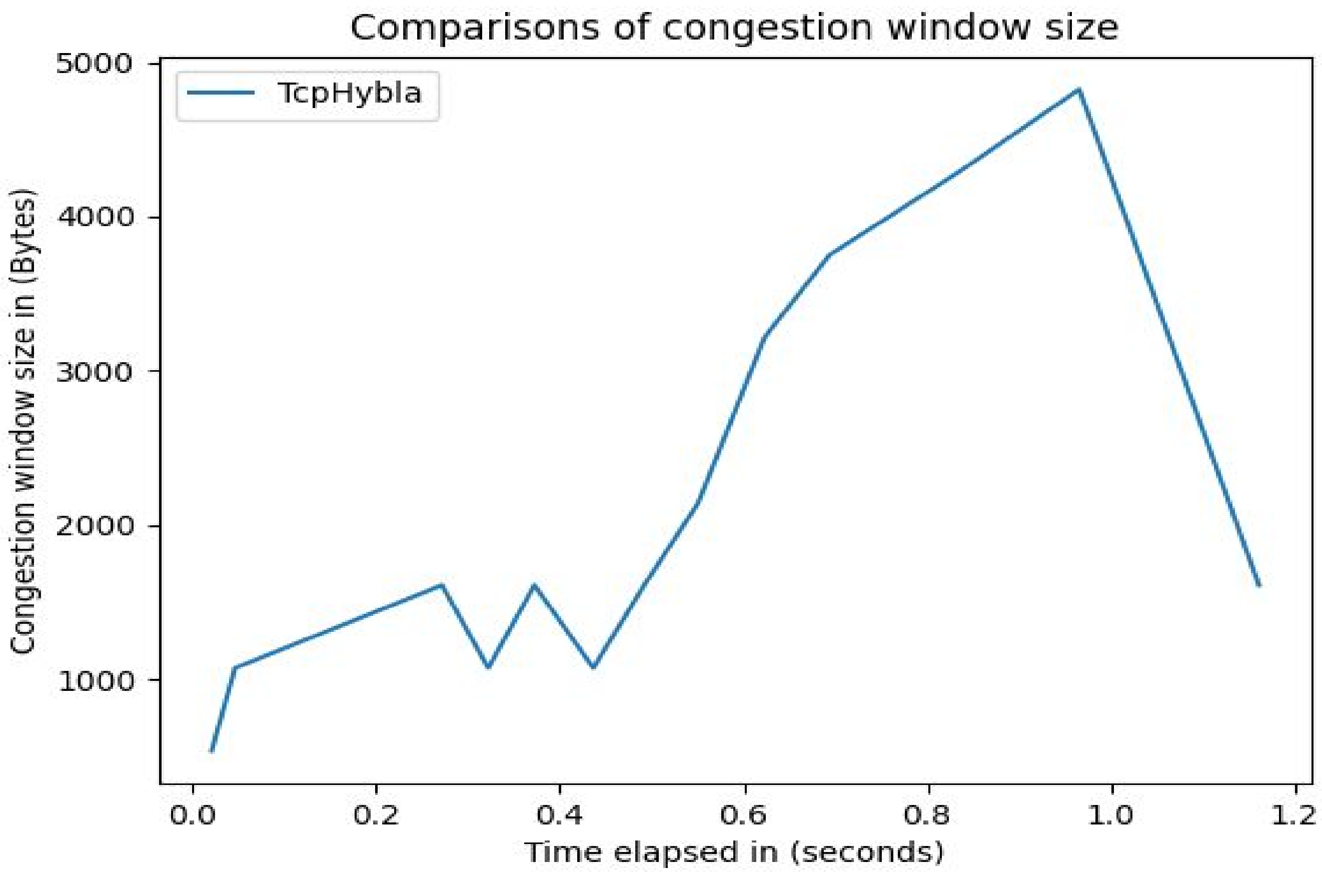
1. TCP New Reno



- **Observation :** We observe in between 0-0.2 seconds, where congestion window size grows exponentially per RTT. From 0.2-0.3 sec we see that the congestion window linearly increases. And about 0.3 sec enters the fast recovery state and window size decreases to its half. After successful fast transmission from 0.5 to 1 sec, it enters into congestion avoidance state and linearly increases thereafter with every ACK response.

- **Inference:** What TCP reno does is, it always tries to keep the transmit window full. Thus for every duplicate ACK received, a new unsent packet is sent. Basically the sender assumes that the corresponding packets might be lost as a result it does not wait for a transmission timeout and directly tries to enter into the fast retransmit phase.

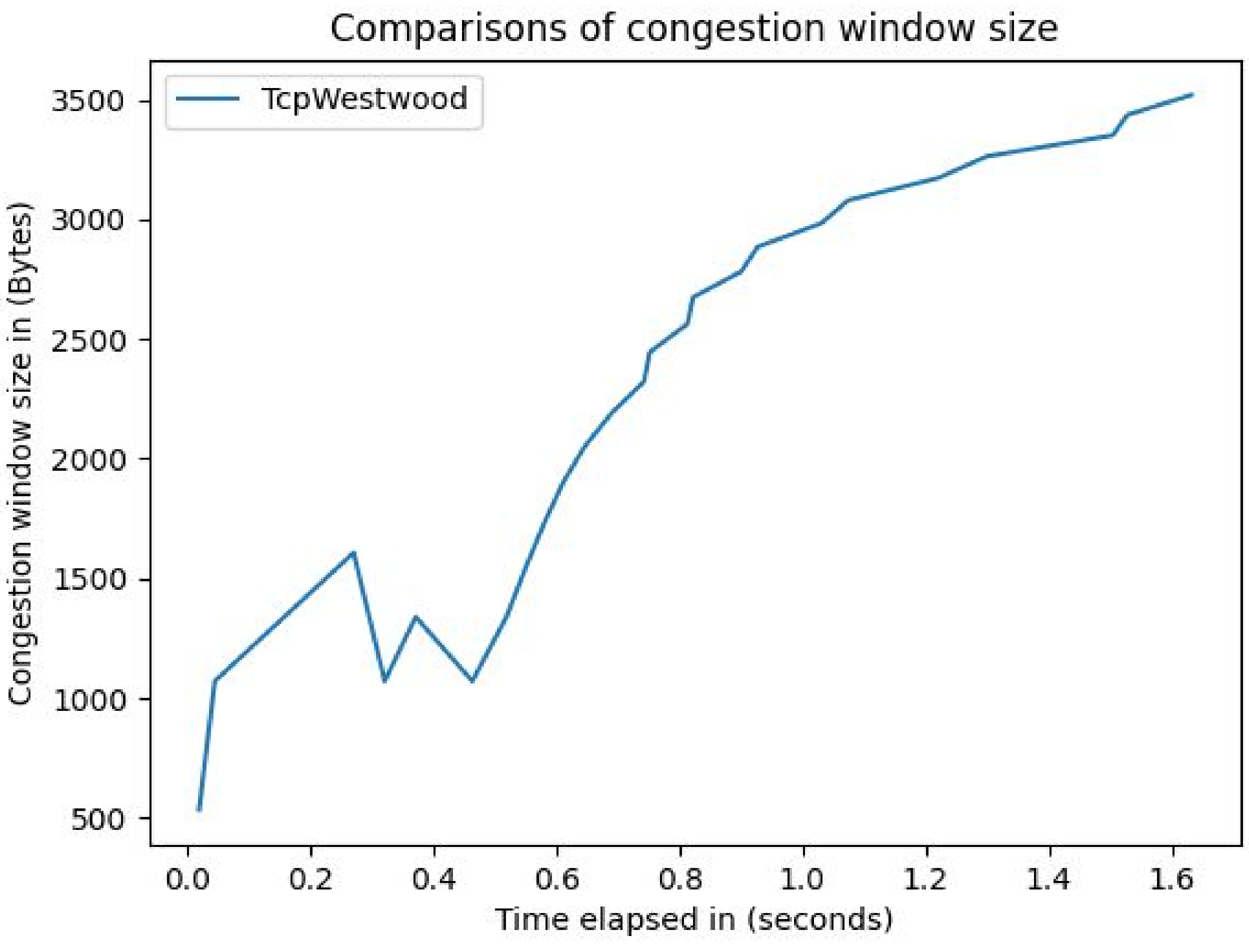
2. TCP Hybla



- **Observation:** Like before, we also see here that the congestion window size increases slowly at the beginning, and then decreases to avoid congestion with fast recovery thereafter. From around 0.6 to 0.9 sec we see linear increase in cwnd size which depicts the congestion avoidance state. At close to 1 sec we see the window size rapidly decreasing entering into fast recovery.

- **Inference:** TCP Hybla uses a congestion window growth formula, where window size increases independent of the Round trip time. As a result there is a large increase in window size incase of larger RTT value. Thus we can say the the throughput in TCP hybla variant is almost independent of RTT value.

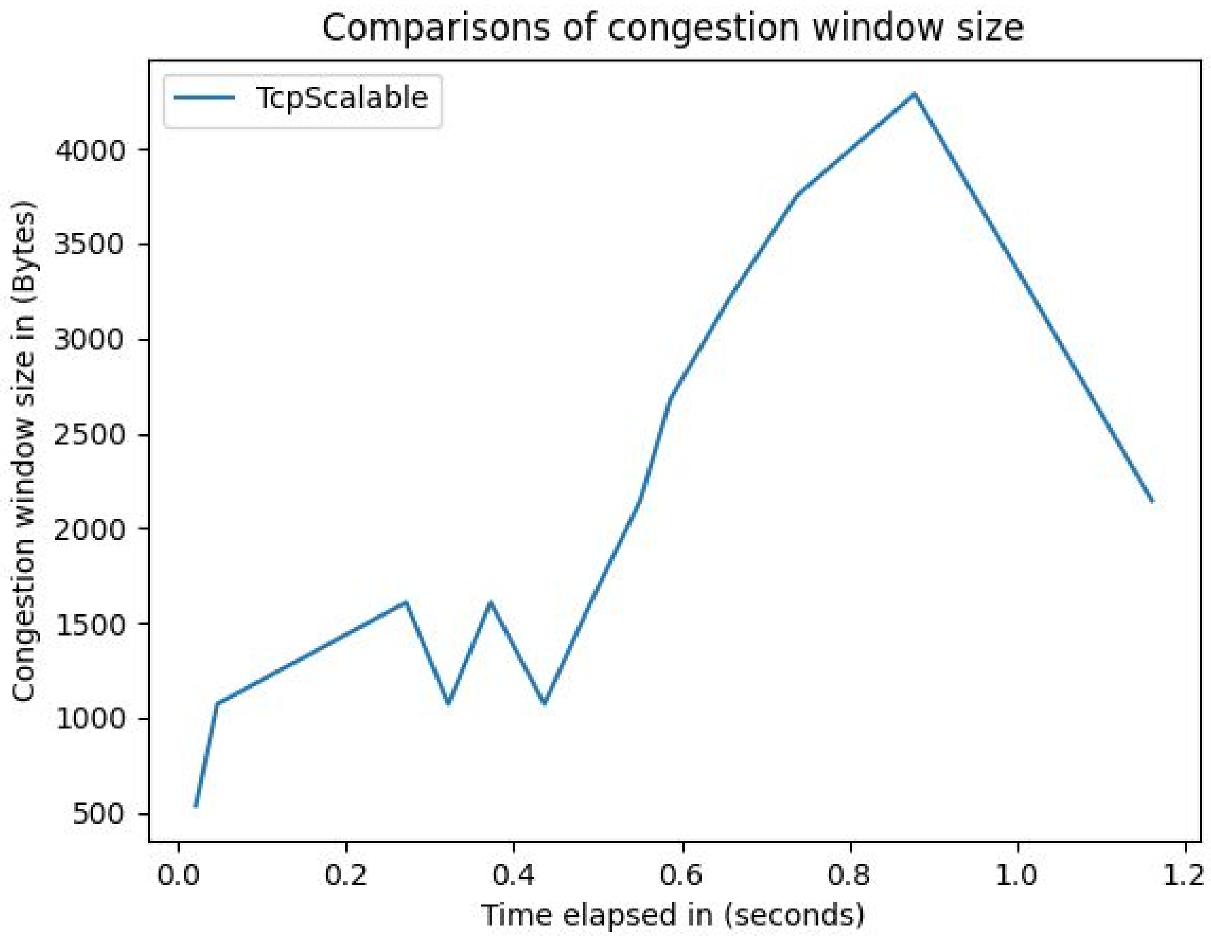
3. TCP WestWood



- **Observation:** The graph for TCP Westwood looks similar to the TCP New Reno state, where we observed slow start state followed by congestion avoidance state of linear increase, which is followed by sharp multiplicative decrease of fact congestion recovery state.

- **Inference:** Westwood is a sender-side-only modification of TCP New Reno variant, intended for links with large bandwidth-delay product. It keeps track of ACK reception, used to estimate "Eligible data rate" and bandwidth for the respective links. It relies on mining ACK stream to assign values to parameters. Due to which it gives efficient performance, without any compromise for fairness.

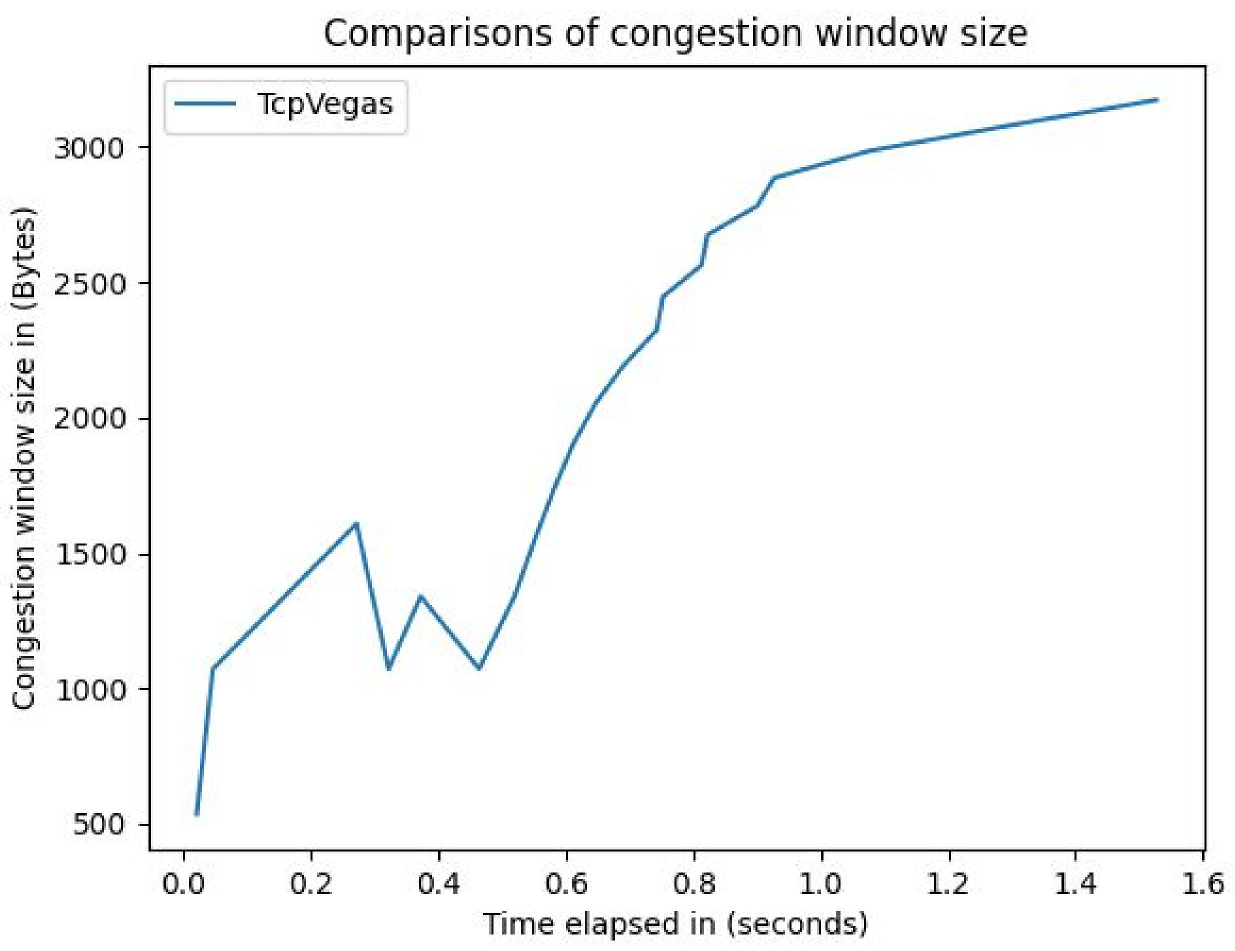
4. TCP Scalable



- **Observation:** It’s trend is quite similar to the Hybla.

- **Inference:**TCP Scalable looks to increase throughput . Instead of halving the congestion window size, each packet loss decreases the congestion window by a factor of 1/8 until packet loss stops then 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. That’s why very large windows take a long time to recover. Leading to fixed increase in cwnd independent of RTT.

5. TCP Vegas

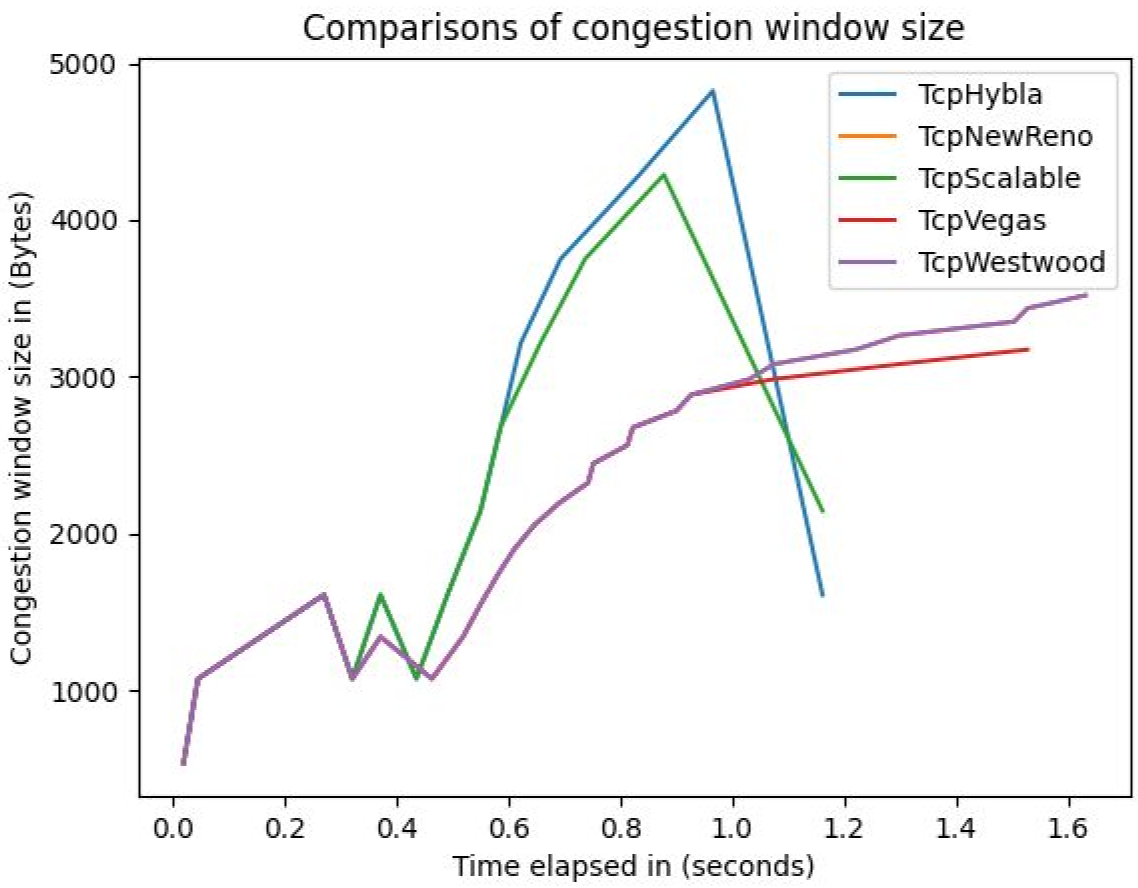


• **Observation:** Similar to previous there is a slow start at the beginning, followed by congestion avoidance and fast recovery. After 0.4 sec, there is a (roughly) linear increase in window size (congestion avoid ancestate). After around 1.0 sec it becomes almost constant.

• **Inference:** To alleviate the effect of ACK compression, Vegas modifies the bandwidth estimation mechanism to perform the sampling every RTT instead of every ACK reception. The result is a more accurate bandwidth measurement that ensures better performance when comparing with Reno or NewReno but still be fair when sharing the network with other TCP connections.

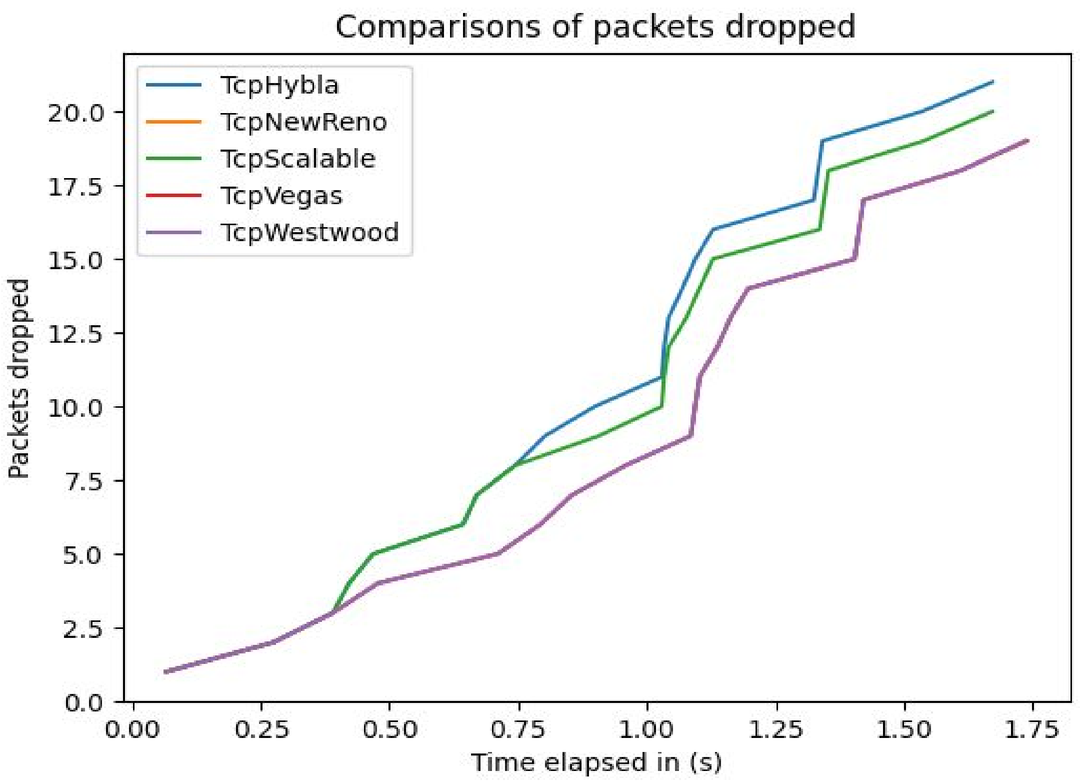
• Vegas compares Expected throughput (window size/RTT) and actual throughput(ACKs/RTT).
-> If actual < expected < actual + α thendecrease cwnd to increase throughput
-> if actual + α < expected < actual + β then do nothing
-> if expected > actual + β then increase cwnd to decrease data rate before packet drop.
(Thresholds of α and β correspond to how many packets Vegas is willing to have in window)

Comparison of congestion window size

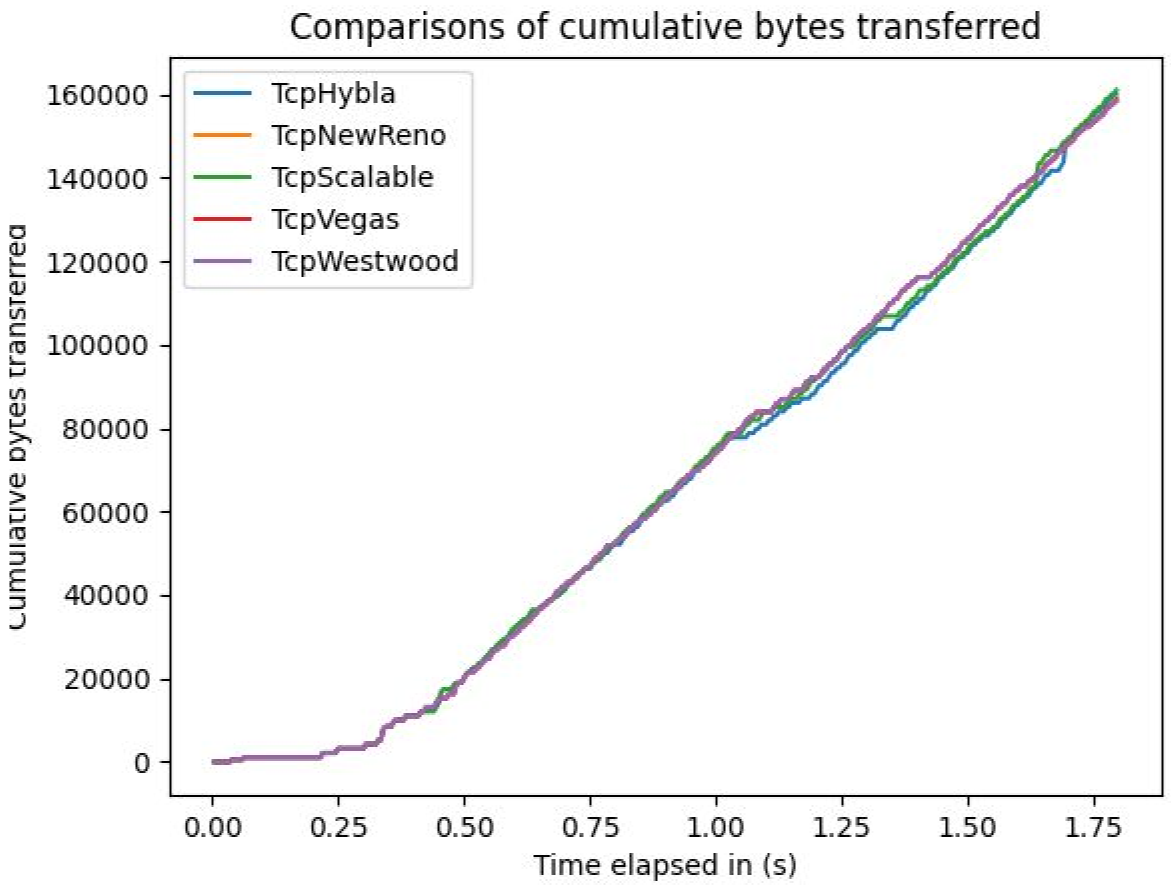


- **Conclusion:** From the above comparison plot, we could readily infer that irrespective of the TCP variant, each of them has the slow start, congestion avoidance, and fast recovery phase respectively. Only what differs is the rate at which the increase and decrease happens respectively, which can be inferred from the slope of the respective plots. While TcpHybla and TcpScalable have more affinity to increase , and multiplicatively decrease. The other three grow at a steady rate, to avoid congestion. Thus, we observe similar trends based on whether congestion window size is changed using packet loss or Round Trip Time delay.

Comparison of cumulative TCP packets dropped



Comparison of cumulative bytes transferred


Comparisons of cumulative bytes transferred
TcpHybla
TcpNewReno
TcpScalable
TcpVegas
TcpWestwood
Cumulative bytes transferred
Time elapsed in (s)