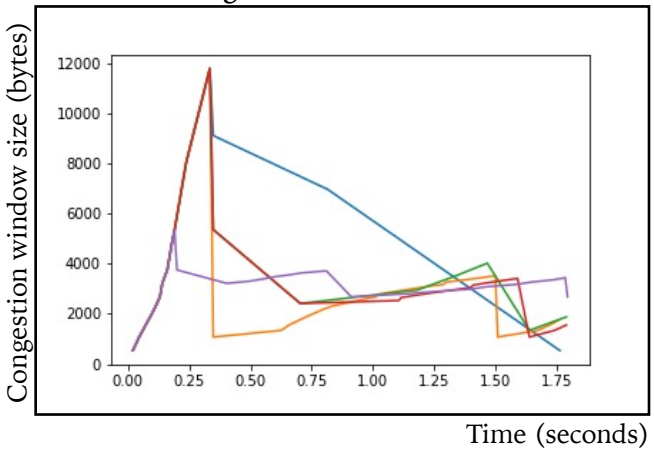
CS342 : Networks Lab, Assignment - 4

Running the program

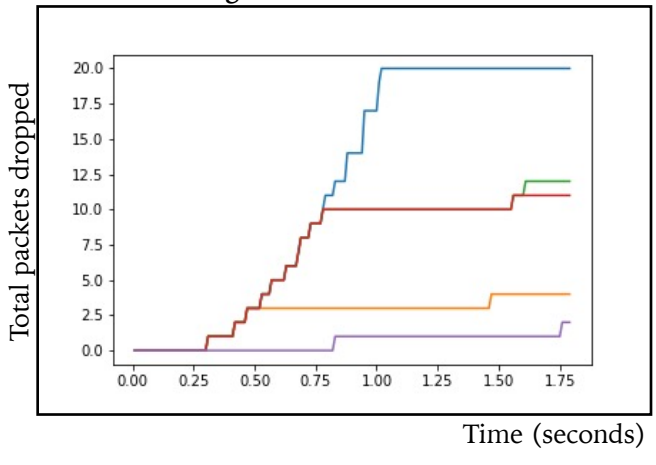
Instructions given as comments in the beginning in sourcecode.cc

Graphs

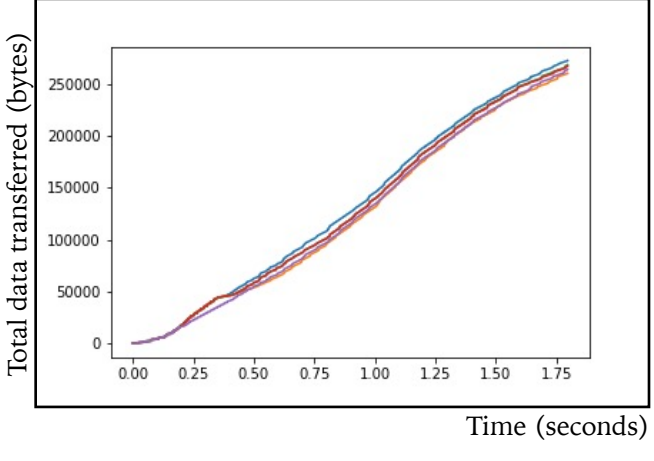
1. Comparing congestion window size amongst various TCP algorithms



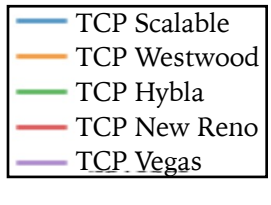
2. Comparing total packets dropped amongst various TCP algorithms



3. Comparing total data transferred amongst various TCP algorithms



Legend (for all three plots)



Observations and inferences

1. TCP New Reno

Initially, there is a slow start phase where the congestion window size increases by *1 byte* for every acknowledgement received (congestion window size increases exponentially with time). Then the network enters a congestion avoidance phase where congestion window size increases linearly with time. Upon experiencing packet losses, the congestion window size is nearly halved, and then halved again until it can go back to sending packets without congestive losses (relieving the congested network). After reaching a steady value, the congestion window increases in size again, until it experiences another period of packet losses after which it follows the previous procedure again.

From the 2nd plot (number of dropped packets v/s time), we see that no packets are lost from roughly $t = 0s$ to $t = 0.3s$. This is the period where congestion window grows initially. After that, there are packet losses until $t = 0.8s$ (approximately) which is the same time duration when the congestion window size is being halved to relieve the network. Then there is another round of the congestion window increasing in size until there are no packet losses (flat line in the plot), and upon experiencing packet losses again ($t = 1.6s$), the congestion window reduces its size. In the 3rd plot (bytes sent v/s time), the bytes-sent-per-time value (slope of the graph), or the throughput, increases steadily in the periods where the congestion window size is increasing and reduces sharply upon experiencing packet losses.

2. TCP Hybla

TCP Hybla has identical values to TCP New Reno in all three plots unto a certain point in time. In the 1st plot, the TCP Hybla congestion window differs from the TCP New Reno congestion window starting from $t = 0.7s$. Initially, it follows the exact same routine as TCP New Reno. After reaching a steady congestion window size value (following the first round of packet losses), TCP Hybla increases its congestion window quicker than TCP New Reno. In TCP New Reno, links with small RTT values grow their congestion window at a higher pace than links with large RTT values. TCP Hybla uses a formula where the increase in congestion window size has less dependence on RTT values than the TCP New Reno formula, and instead of increasing the congestion window by *1 byte* for every ACK received, it increases it by ρ^2 *bytes* (per ACK) , where ρ is the ratio of RTTs (if ρ is less than *1*, then it increases by *1 byte* per ACK so that links with small RTTs are not penalized). This explains the faster growth of the TCP Hybla congestion window in the second half of the experiment.

Because of the initial similarity between TCP Hybla and TCP New Reno, they have the same values in the 2nd plot. Only towards the far end of the experiment, TCP Hybla has more packet losses because it grew its congestion window faster than TCP New Reno. This also explains why their 3rd plot lines are overlapping during the entire experiment, except for the final phase.

3. TCP Westwood

TCP Westwood is a sender-side modification of TCP New Reno. TCP Westwood uses bandwidth estimation and continuously monitors the ACK reception rate. This gives rise to differences between TCP Westwood and other TCP variants, especially in links with large bandwidth-delay products. The bandwidth estimation allows the sender to alter the congestion window size as per the conditions in a better manner. As evident in the 1st plot, when the network faces packet losses, TCP Westwood is the quickest to reduce its congestion window size to a steady value. Other TCP variants use a more trial-and-error based strategy to reduce their congestion window sizes (halving it until there are no packet losses), whereas TCP Westwood has a more deterministic approach to resize the congestion window (using bandwidth estimation). After adjusting its congestion window size quickly, it also grows it quicker than other variants (observing the slope of the congestion window size v/s time plot).

Because of a better way to change the congestion window size, TCP Westwood drops less packets than most other variants, as shown in the 2nd plot. TCP Westwood uses a smaller congestion window in most parts of the experiment, and thus transfers less number of bytes (over the entire duration of the experiment).

4. TCP Scalable

TCP Scalable is generally very efficient over large networks. It increases and decreases its congestion window sized at fixed rates. After facing packet losses, TCP Scalable reduces its congestion window size by much smaller factors than other TCP variants ie. for $cwnd_{new} = \beta \cdot cwnd_{old}$, the β value is quite large (usually 0.875). It starts off similar to other variants, as shown in the 1st plot. After facing packet losses, TCP Scalable reduces its congestion window size very gradually, keeping it much higher than other variants' congestion window sizes for most part of the experiment.

Because of a large congestion window, TCP Scalable has the highest number of dropped packets over the course of the experiment. Because of TCP Scalable's preference to keep operating with a large congestion window, it takes longer to reach a steady congestion window size (where packet loss does not occur), and thus it continues experiencing packet losses for a longer duration. Moreover, the large congestion window also allows TCP Scalable to transmit the most number of bytes in the entire experiment. At every point in time, TCP Scalable has the highest value of bytes sent (in the 3rd plot).

5. TCP Vegas

TCP Vegas implements a vastly different congestion control mechanism than other TCP variants. Other TCP variants decrease their congestion window size upon facing packet losses, whereas TCP Vegas uses a very cautious approach towards this by analyzing RTT values to decrease its congestion window size. TCP Vegas continuously keeps a check on RTT values and anticipates any impending packet loss situation. As soon as the throughput (calculated using number of bytes sent and RTT) drops to a certain value (throughput decreases because of increased RTT, which occurs because of congestion in the network), TCP Vegas reduces it congestion window size, throttling the link, and returns to a safe state (with better throughput). It increases its congestion window size again, before proactively determining if it would face a packet loss situation if it continues in the same manner. Every time TCP Vegas predicts that the network would be congested in the near future, it reduces its congestion window size. As a result of This, TCP Vegas has the smallest congestion window for majority of the experiment.

Since other TCP variants reduce their congestion windows only after losing packets, TCP Vegas drops the fewest number of packets throughout the experiment (takes control measure much before facing packet loss), as clearly evident in the 2nd plot. However, as a downside of using the smallest congestion window, TCP Vegas sends the least number of bytes in the experiment (shown in the 3rd plot).