COL334 Assignment 3

Aniruddha Deb
2020CS10869

November 2022

Task 1

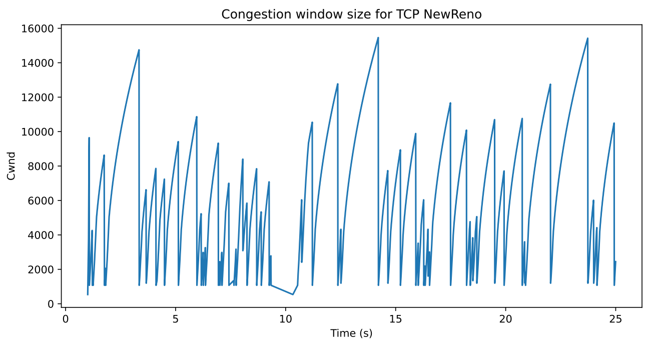
For task one, the `seventh.cc` file was modified with minor changes, to set the TCP connection type at the start. The congestion window data is logged to a file using a AsciiTraceHelper and the pcap files are generated using a PcapHelper. All other parts were unchanged.

1. The table is as follows:

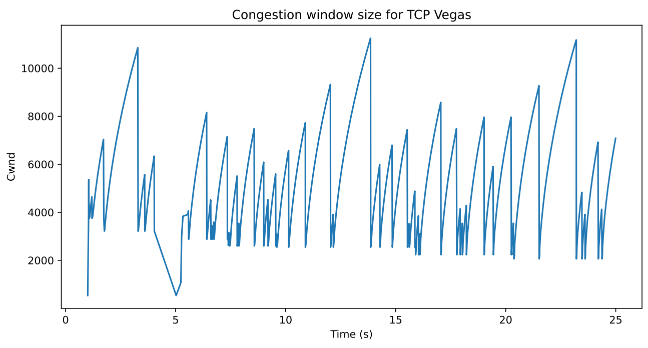
Protocol name	Max. Window Size	No. Packets dropped
NewReno	15462	57
Vegas	11252	58
Veno	15462	58
Westwood	15471	59

2. The graphs of congestion window size v/s time are as follows:

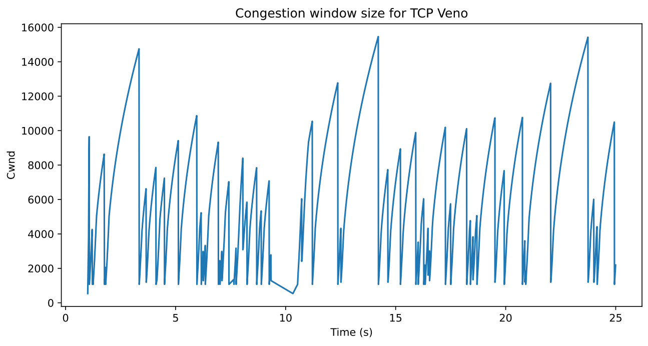
   (a) NewReno enters retransmission whenever it receives three duplicate acknowledgements, but it doesn't exit until all the unacknowledged packets have been sent. It follows the same basic principles as Reno (AIMD, slow start).



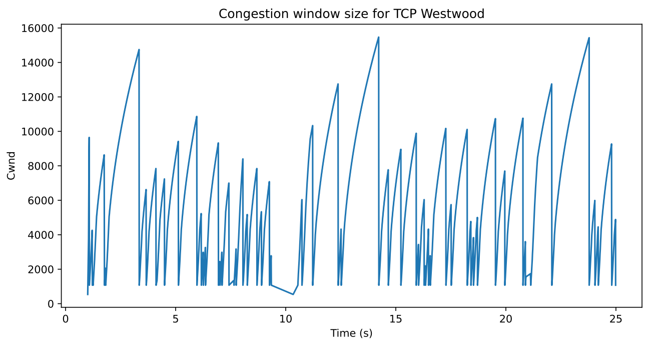
   (b) Unlike the others relying on packet drops, Vegas relies on RTT time to change it's congestion window. It still follows AIMD and slow start, but it adds a new retransmission mechanism. Whenever a duplicate acknowledgement is received, it checks if more than RTT time has elapsed since the segment was transmitted, and if so, retransmits without waiting for duplicate acks. Hence, the behaviour around 5s is it's slow start, and all others are predicted packet drops, allowing it to recover faster.



(c) Veno is a combination of Vegas and Reno: while Vegas tries to keep the backlog at the bottleneck queue (estimated as a difference between expected and actual throughput, multiplied by base RTT) less than a constant by adjusting the congestion window size, Veno will increase the congestion window by one after two round-trip times rather than once when the backlog is more than the constant.



(d) Westwood is a modification of TCP NewReno that allows it to handle large pipes (lines which take a long time to transmit). It mines the acknowledgement stream for information which helps it set the congestion control parameters



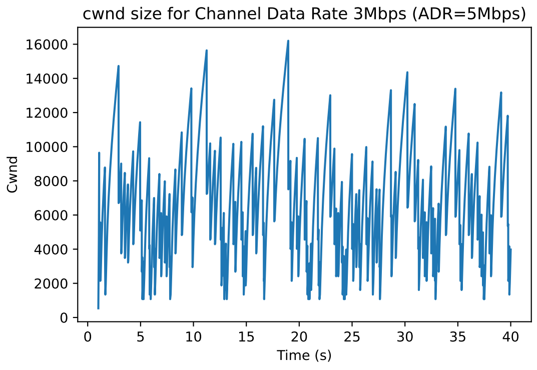
3. All the three graphs (except TCP Vegas) have similar behaviour, as all three are minor variations of the same underlying principles (AIMD and slow start), and in the absence of other nodes transmitting packets on the network, we can't see major differences between them. Most minute differences originate due to the stochasticity of the error rates.

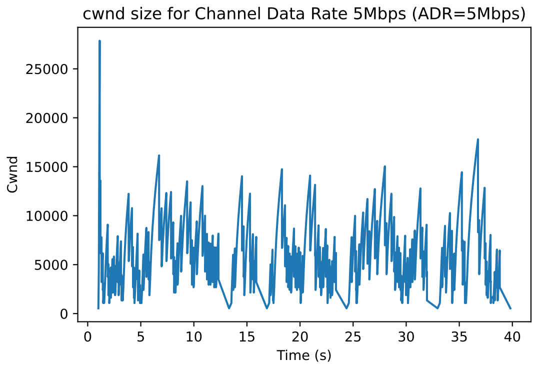
4. BBR, unlike other congestion control algorithms, does not use the assumption that packet loss equals congestion. Instead, it uses Latency to decide whether the network is congested or not. BBR, via it's congestion window control, also attempts to not fill the buffers and performs much better than other algorithms when it comes to long distance packet transmission.

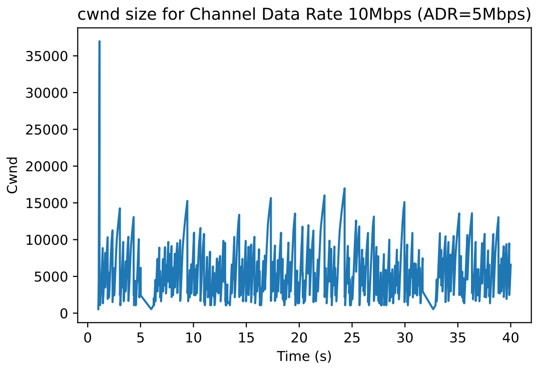
Task 2

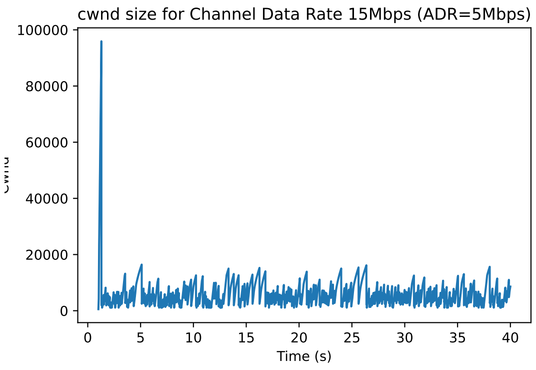
For Task 2, `seventh.cc` was modified, but the entire simulation was placed in a function, which was called repeatedly in the main function with different application and channel data rates. Not many changes were done to the simulation itself, and the congestion control model was fixed as TCP NewReno

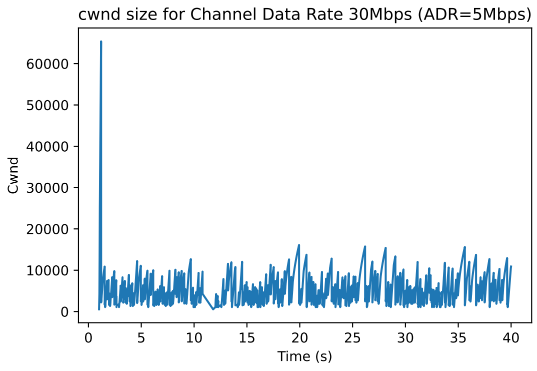
Keeping Application data rate constant:



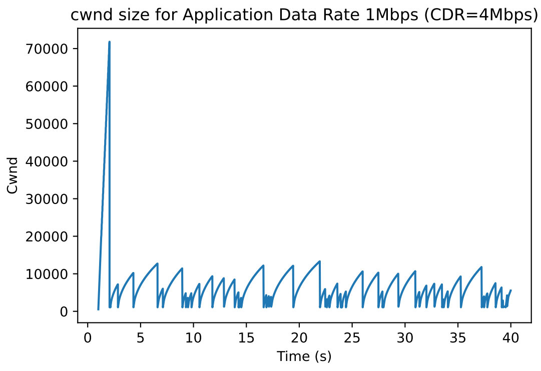


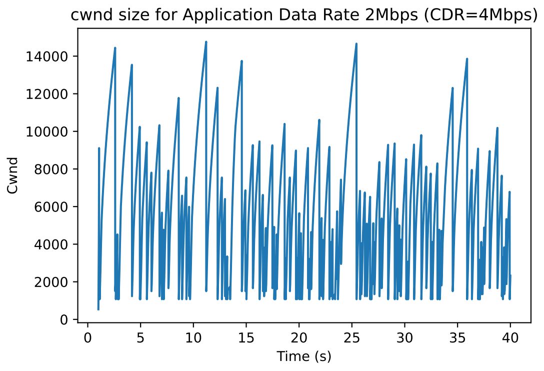


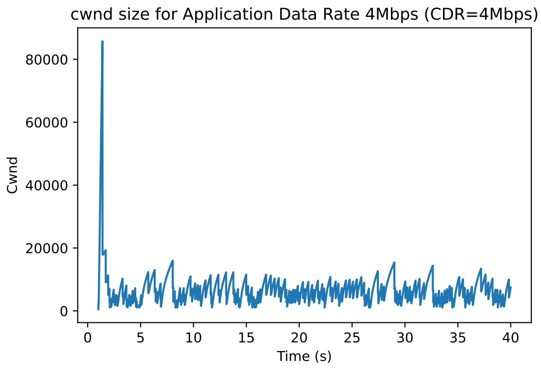


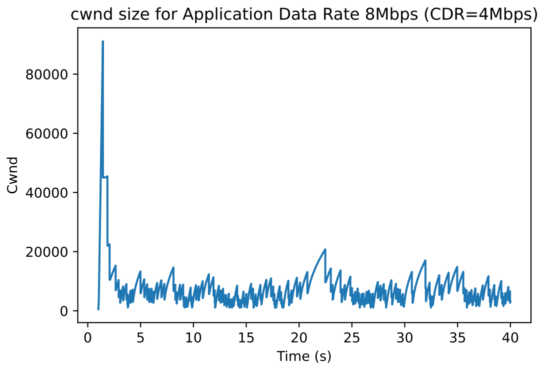


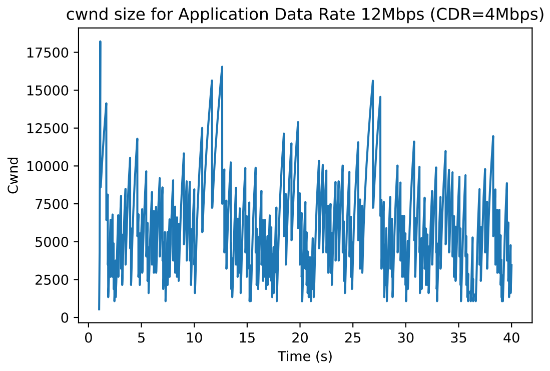
Keeping Channel Data rate constant:











We notice that:

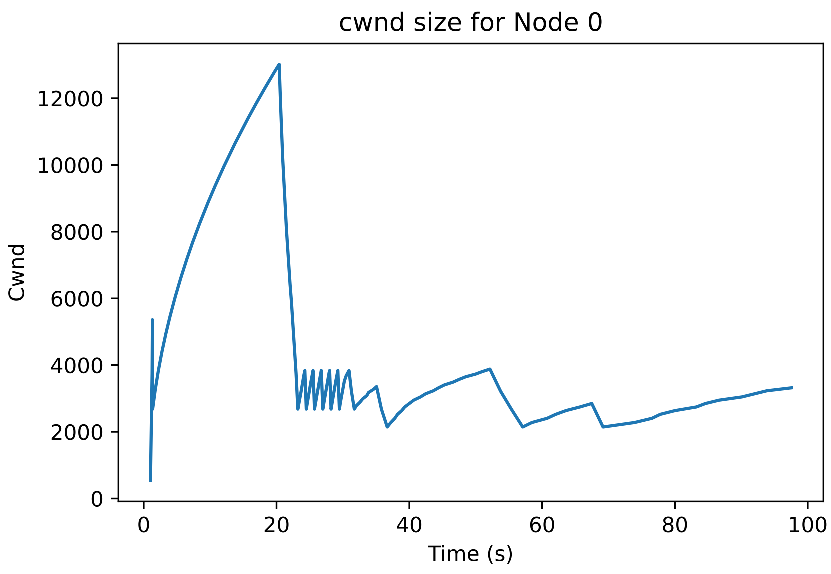
- Increasing the application data rate (while keeping channel data rate constant) results in more frequent timeouts/packet drops, as a result of which the congestion window changes. The average congestion window decreases as the application data rate increases
- Increasing the channel data rate (while keeping application data rate constant) results in less frequent packet drops, as there is more channel bandwidth to utilize. The average congestion window size increases as the application data rate increases.
- The channel data rate serves as a bottleneck to the application data rate: with a fixed channel data rate, no matter the application data rate, only so much data can pass through the channel. This results in a large number of packet drops if the application data rate overwhelms the channel data rate.

Task 3

For task 3, the network we use had to be implemented from scratch: for this, the steps that were taken were:

1. The nodes are first constructed in a NodeContainer
2. The internet stack is then installed on the nodes
3. The four p2p channels between the nodes are created
4. The Error model is installed on the NetDevices of the nodes
5. IP addresses are assigned to the four NetDeviceContainers created after creating p2p channels
6. The routing tables are populated to ensure that the packets are forwarded to the correct nodes
7. TCP and UDP sink addresses are created, and Sink Helpers are installed on the target nodes
8. The TCP and UDP sockets are created, and then the transmitting apps are created and installed in the internet stack of the transmitting nodes
9. The loggers are attached to the nodes (AsciiTraceHelper and PcapHelper)
10. The simulation is started.

The graph and explanations are as follows:

1. 

2. We can see that the UDP application was turned on at 20 seconds, as packet drops become more frequent. Since vegas changes it's congestion window based on RTT time, such that the estimated backlog is less than a constant (as discussed above), at 30 seconds, when the network is overloaded with UDP packets, Vegas increases the congestion window more slowly to compensate for the overloaded network and packet drops, and this is noticeable in the graph that has been plotted.

3. The pcap files containing transmitted packets and packet drops is attached with the submission, in the Q3 directory.

Appendix

The files I've submitted are the source files (`a3q1.cc`, `a3q2.cc` and `a3q3.cc`). All three generate .cwnd files containing congestion window information, and a1 and a3 files generate a pcap file containing information regarding dropped packets.

For plotting, python was used. The jupyter notebook (or converted python script), either can be run from the base directory to generate the plots used in the corresponding directories.