Matt Wong, mwong82, Fall 2020

Project 5 Analysis

1. Item #1
   a. I would expect the initial part of the curve to look concave, or sharply increasing over time. Once congestion occurs, CUBIC will still seek to maximize throughput and achieve a throughput similar to Reno, using a multiplicative decrease approach. This will result in a sharp sawtooth pattern.
2. Item #2
   a. As previously mentioned, knowing that TCP Cubic follows an exponential increase in the congestion window until experiencing congestion events, this is why I expect to see an initial concave pattern. Also knowing that CUBIC will follow a multiplicative decrease while keeping track of the previous point of congestion and repeating the process, it would be expected to see a sawtooth pattern with respect to the congestion window over time.
3. Item #3
   a. With respect to queue occupancy, it is obvious that the larger queue displays a higher packet count on its y-axis, because the number of packets in the queue is greater. The behavior is the same, yet the frequency that the the algorithm cycles through a congested vs. uncongested state is greater with the smaller queue; these differences occur because the determined wmax is much greater (in the small queue, it is roughly 30, and in the larger queue it is 150), so it will take longer to reach the congestion threshold in the larger queue, resulting in a less frequency of cycling through congested and uncongested states.

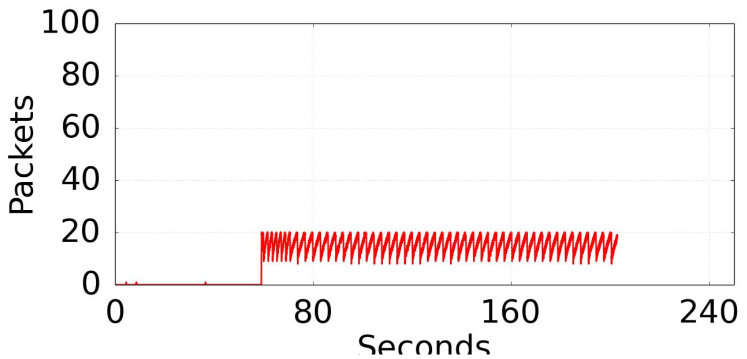


Figure 1. Small queue of packet congestion over time for TCP Reno.

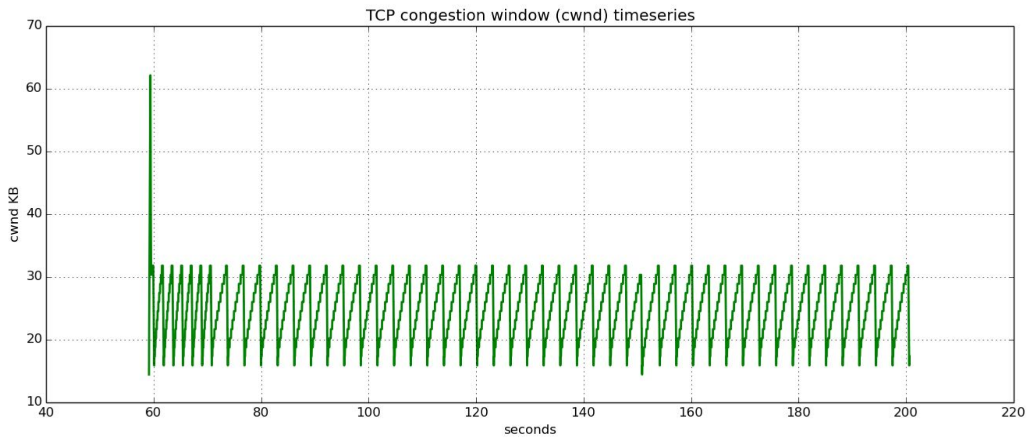


Figure 2. Small queue mapping the congestion window over time for TCP Reno.

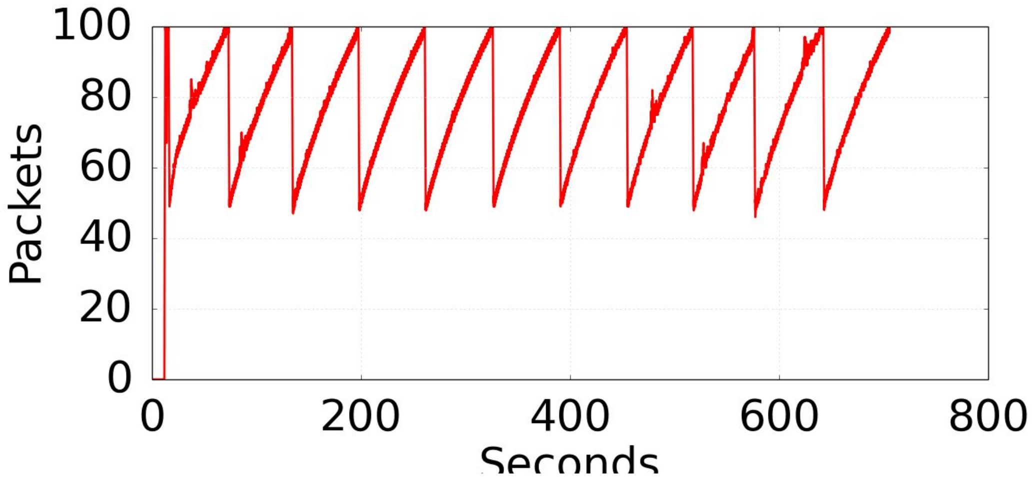


Figure 3. Large queue of packet congestion over time for TCP Reno.

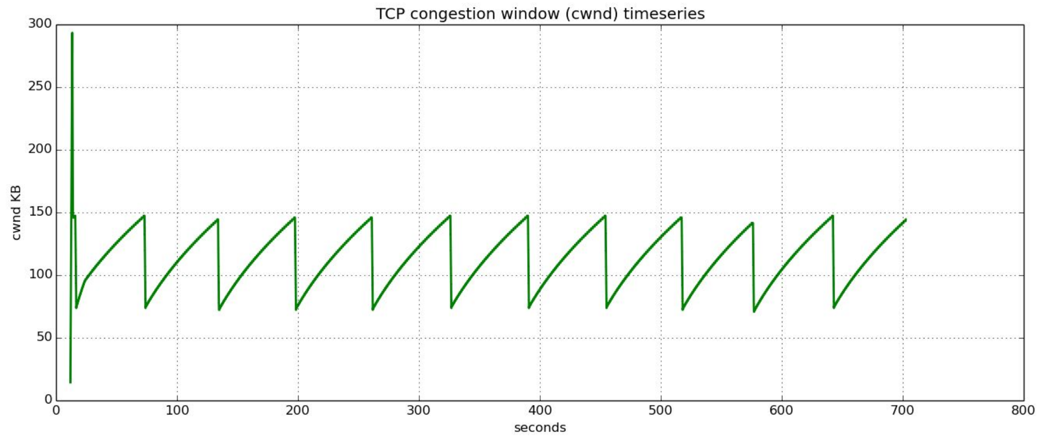


Figure 4. Large queue mapping the congestion window over time for TCP Reno.

4. Item #4
   a. The graphs below show that the congestion window includes a very steep increase initially followed by varying multiplicative decreases during period of congestion. This is likely because this decrease is also based on a constant, and the wmax congestion threshold is varying over time. TCP cubic increases the congestion window sharply and then slowly increases it as it approaches the maximum threshold.

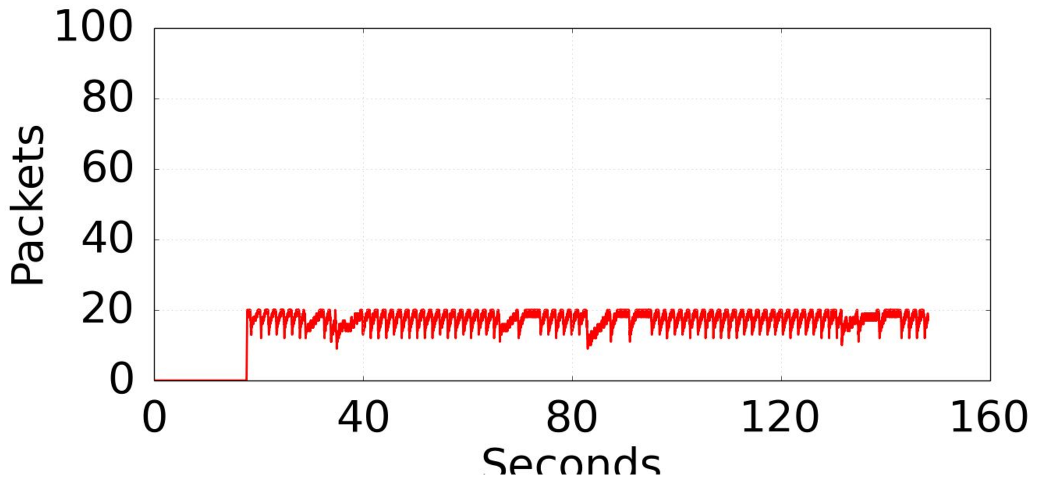


Figure 5. Small queue of packet congestion over time for TCP Cubic.

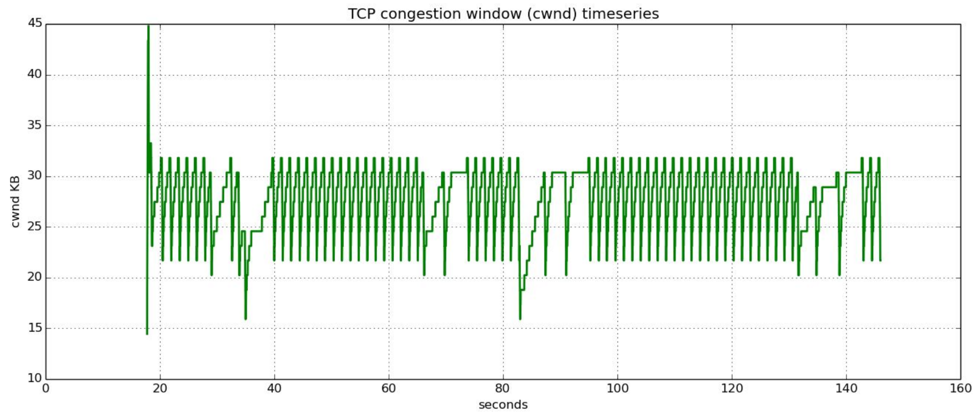


Figure 5. Small queue mapping the congestion window over time for TCP Cubic.

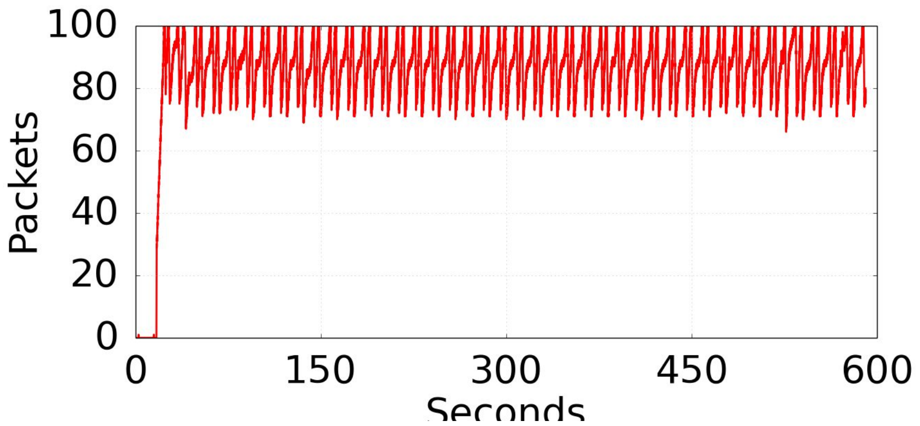


Figure 6. Large queue of packet congestion over time for TCP Cubic.

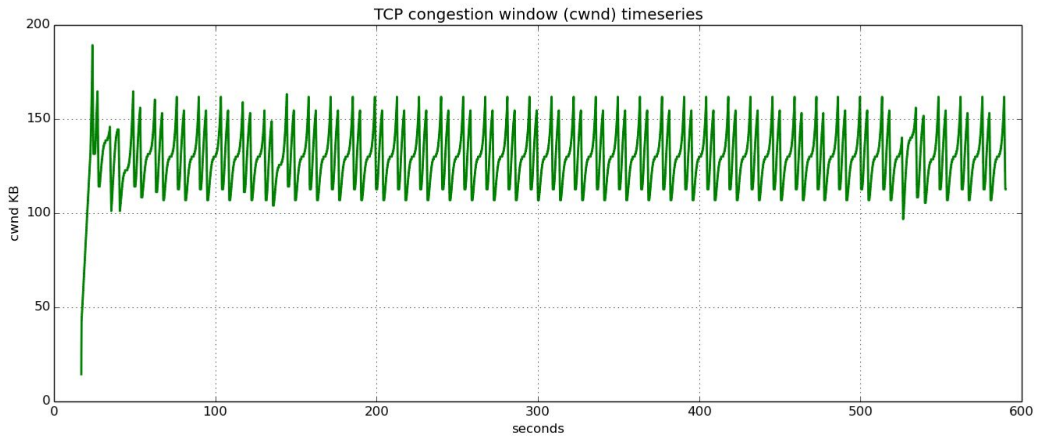
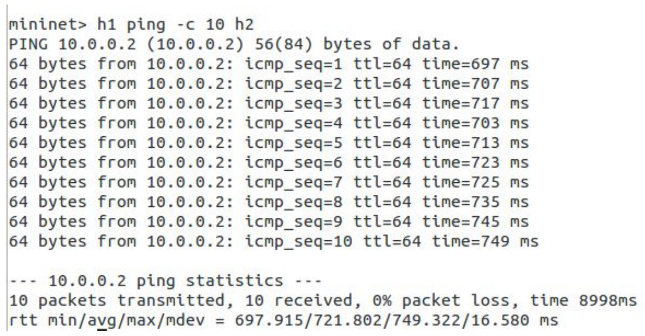


Figure 7. Large queue mapping the congestion window over time for TCP Cubic.

5. Item #5

   a. The results were slightly different than the prediction; I was expecting more of a concave shape initially, but because of the sizing on the graph, it is more of a sharp jump up to the initial maximum congestion threshold (straight line up). The larger queue seemed to match up more with the prediction, and this makes sense, as TCP Cubic is designed for maximizing bandwidth over long distance, high latency networks.

6. Item #6
   a. The presence of a long lived flow sharply increases the download speed (by about 35x), or increases the overall RTT relative to a situation where there are not multiple competing TCP flows. This is because multiple flows are contending for bandwidth on the network, and factors such as the RTT of each individual flow, network connection strength, and more.



```
mininet> h1 ping -c 10 h2
PING 10.0.0.2 (10.0.0.2) 56(84) bytes of data.
64 bytes from 10.0.0.2: icmp_seq=1 ttl=64 time=697 ms
64 bytes from 10.0.0.2: icmp_seq=2 ttl=64 time=707 ms
64 bytes from 10.0.0.2: icmp_seq=3 ttl=64 time=717 ms
64 bytes from 10.0.0.2: icmp_seq=4 ttl=64 time=703 ms
64 bytes from 10.0.0.2: icmp_seq=5 ttl=64 time=713 ms
64 bytes from 10.0.0.2: icmp_seq=6 ttl=64 time=723 ms
64 bytes from 10.0.0.2: icmp_seq=7 ttl=64 time=725 ms
64 bytes from 10.0.0.2: icmp_seq=8 ttl=64 time=735 ms
64 bytes from 10.0.0.2: icmp_seq=9 ttl=64 time=745 ms
64 bytes from 10.0.0.2: icmp_seq=10 ttl=64 time=749 ms

--- 10.0.0.2 ping statistics ---
10 packets transmitted, 10 received, 0% packet loss, time 8998ms
rtt min/avg/max/mdev = 697.915/721.802/749.322/16.580 ms
```

7. Item #7
   a. The performance of the download is better with a smaller queue size versus a larger queue size, and as a result reduces the download time for wget. This is because in the case of a larger queue, the server is allocating, or buffering more memory, even if unnecessary. This results in a longer delay, and ultimately bloats the buffer, delaying short flow packets from reaching their destination.

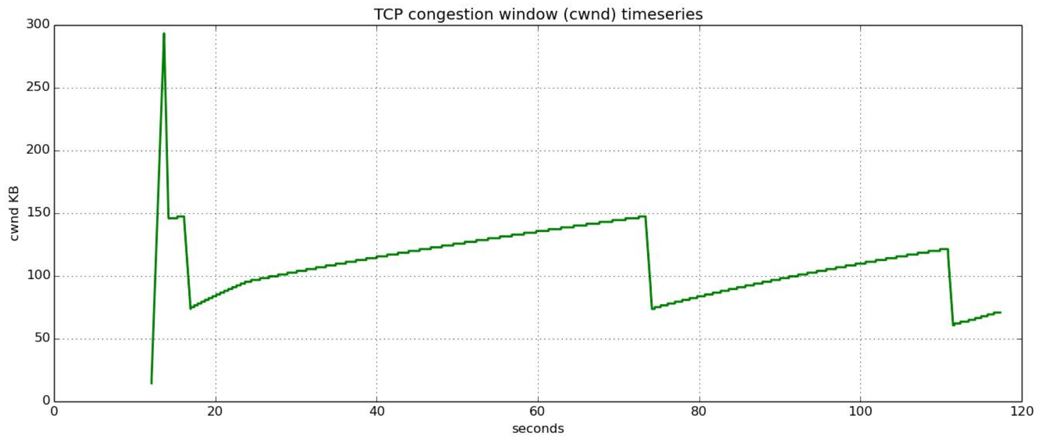


Figure 8. Large queue mapping the congestion window over time for experiment-1.

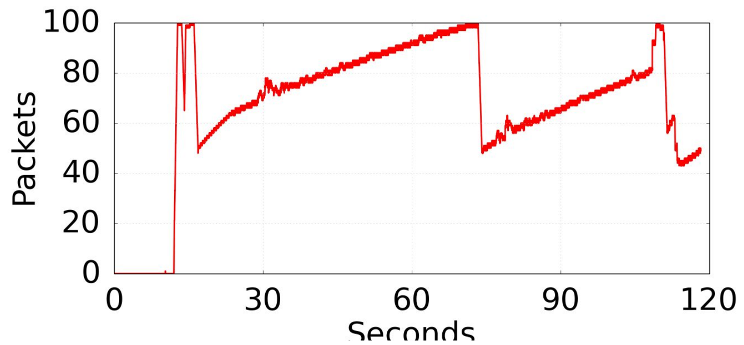


Figure 9. Large queue measuring packet congestion over time for experiment-1.

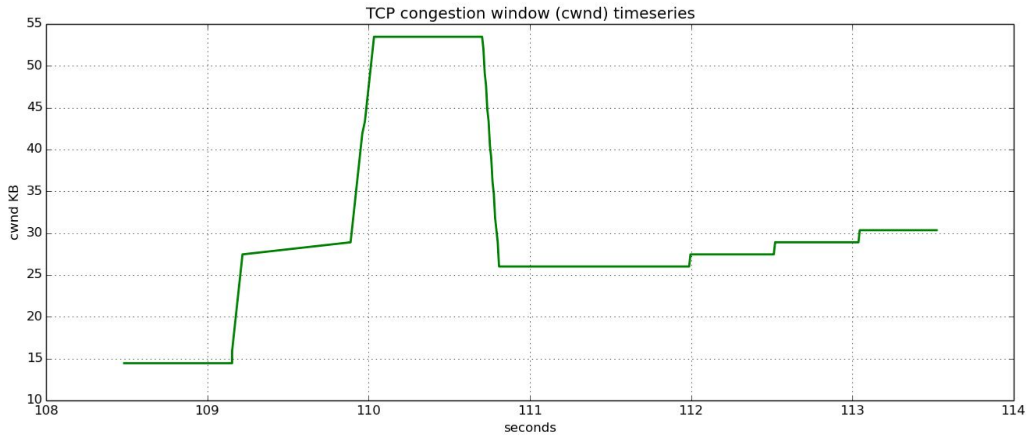


Figure 10. wget cwnd over time for experiment-1.

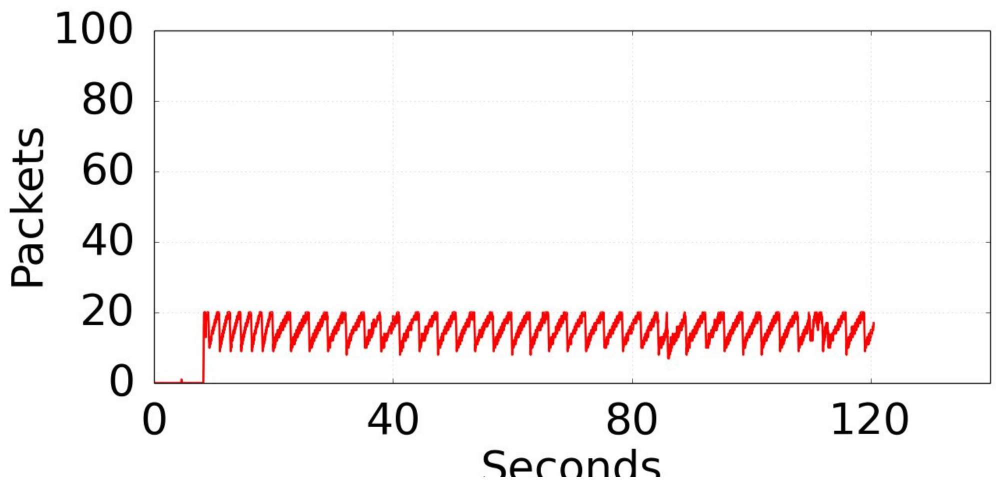


Figure 11. Small queue measuring packet congestion over time for experiment-2.

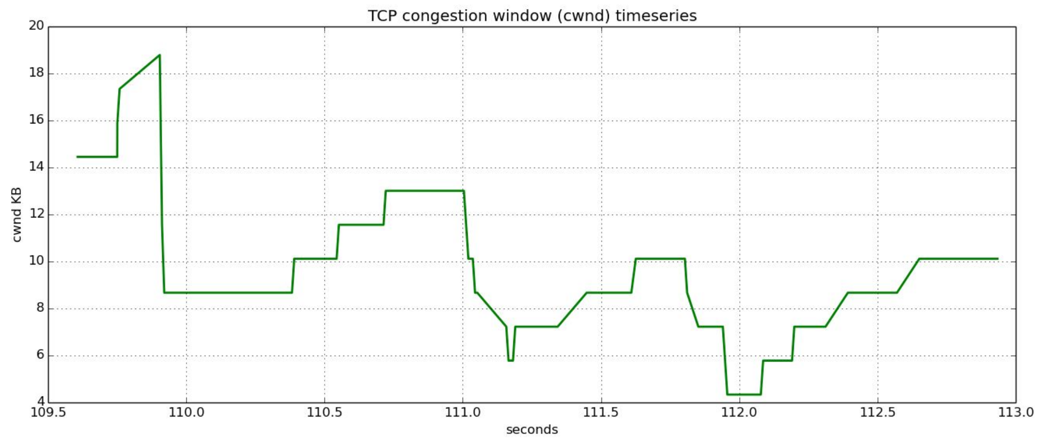


Figure 12. wget cwnd over time for experiment-2.

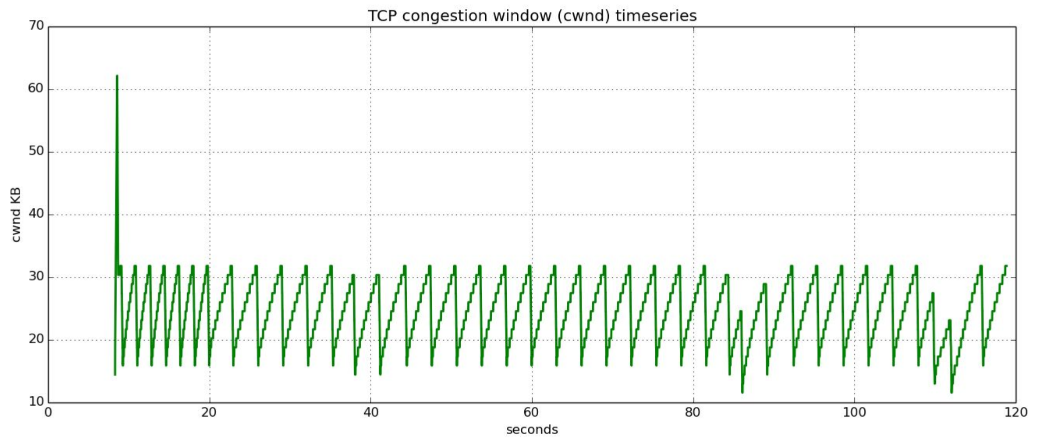


Figure 13. Small queue mapping the congestion window over time for experiment-2.

8. Item #8
    a. The presence of a long lived flow on the link when using two queues has no effect on the RTT, or latency, as without the flow, the RTT was:
    rtt min/avg/max/mdev = 20.178/20.270/20.619/0.176 ms
    And with the long-lived flow, the latency was nearly identical:
    rtt min/avg/max/mdev = 20.101/20.475/22.360/0.708 ms
    The download speed was significantly faster than with a single queue.
    87.3KB/s in 2.0s
    And this is expected, as the processing occurs separately, and there is proper traffic control imn place to prevent the previously seen buffer bloat.

9. Item #9
    a. It was observed that there was a slight improvement in overall throughput and TCP fairness when observing multiple flows competing for bandwidth under the TCP CUBIC algorithm vs. the TCP Reno algorithm. This could be observed firstly

through an overall higher average cwnd for wget over time, meaning that more was allowed to be sent into the network before receiving an ACK. In addition to the observed difference in cwnd for wget, the overall network cwnd was higher, steadily between 100-150 in TCP CUBIC, but in TCP Reno, this falls below 100 KB (experiment-1 vs. experiment-3). TCP CUBIC tends to be more fair regardless of individual flow RTT, offering the opportunity to steal more bandwidth when necessary, as well as adjusting the congestion window based on the cubic function, and I think this greatly contributes to the overall difference in throughput. Knowing that TCP CUBIC is the standard in Linux operating systems, I am not surprised to see this outperformance by TCP CUBIC, especially in the case when we are testing multiple flows with potential opportunities of high latency.

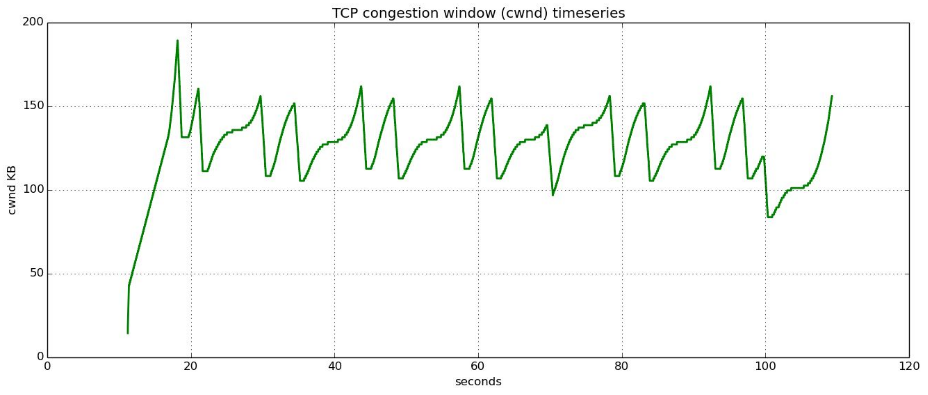


Figure 14. Large queue mapping the congestion window over time for experiment-3.

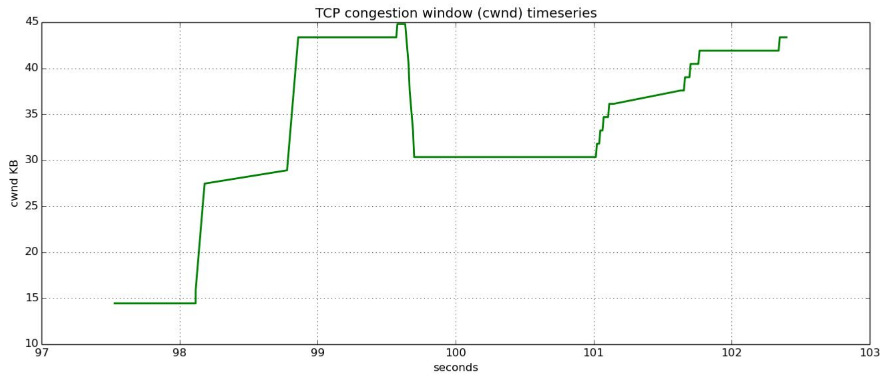


Figure 15. wget cwnd over time for experiment-3.

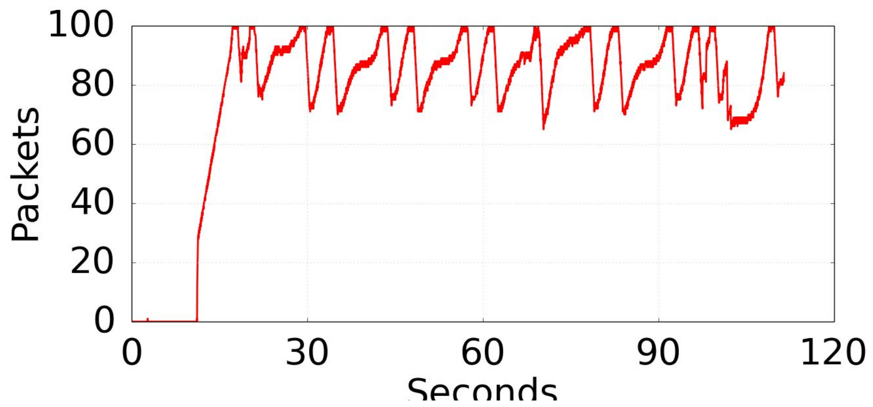


Figure 16. Large queue measuring packet congestion over time for experiment-3.

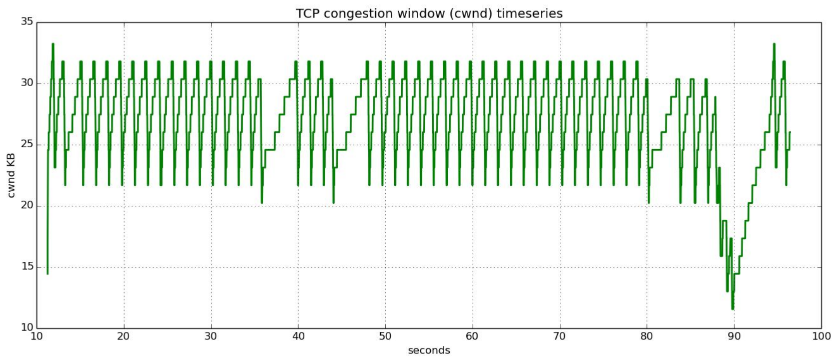


Figure 17. Small queue mapping the congestion window over time for experiment-4.

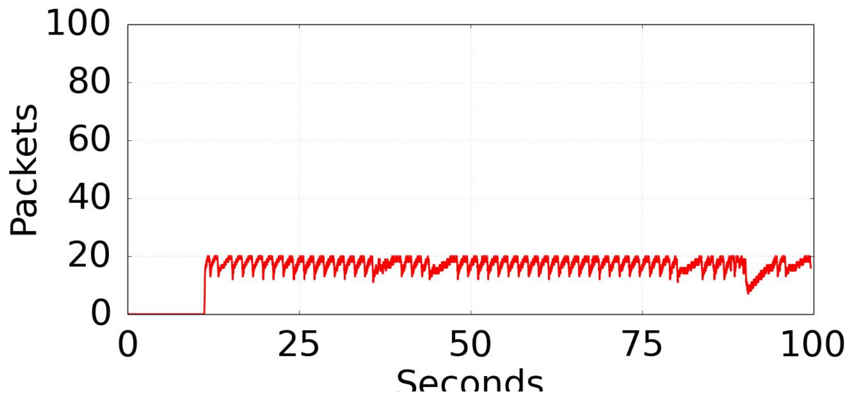


Figure 18. Small queue measuring packet congestion over time for experiment-4.

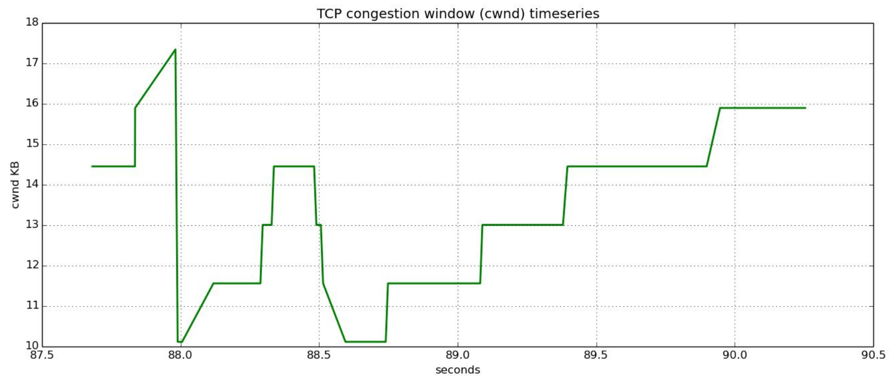


Figure 19. wget cwnd over time for experiment-4.