Project 5: Congestion Control & Buffer Bloat
Hans Lau
CS6250 OMSCS

Before the experiment

Item #1: In this network, what do you expect the CWND (congestion window) curve for a long lived TCP flow using TCP CUBIC to look like? Contrast it to what you expect the CWND for TCP Reno will look like.

Item #2: Explain why you expect to see this; specifically relate your answers to details in the paper. (We expect you to demonstrate the ability to apply what you have already learned from the paper in your answer to this question, and that you've furthermore thought about what you read in order to make a prediction. So justifying your prediction is important. — It's okay if your prediction ends up being wrong as long as it was based on an understanding and analysis of the paper!)

According to the equation,

$$W(t) = C(t - K)^3 + Wmax \text{ where } K = \sqrt[3]{\frac{Wmax\,\beta}{C}}$$

the CWND for a long-lived TCP flow using TCP CUBIC will contain concave and convex regions. Specifically, after a loss event, Wmax is set to be the window size where the loss event occurred and performs a multiplicative decrease with a factor of beta (window decrease constant). Then in the congestion avoidance phase, it grows in a concave fashion till Wmax is reached or no packet is dropped. Once it reaches Wmax, it continues to grow, but now in a convex fashion due to the cubic nature of the equation for W(t). The exponential growth continues till there is a packet loss. In the entire duration, whenever there is a packet loss, that size of the window is marked as Wmax and previous Wmax value is stored in Wlast_max. Thus, if network bandwidth per flow remains constant, then only concave region of the cubic growth function would appear along with the plateau near Wmax. If network bandwidth per flow increases, the graph will include both concave and convex curve with Wmax at the tipping point. If network bandwidth per flow decreases, only a part of the concave curve will be seen and the plateau may not be reached around Wmax.

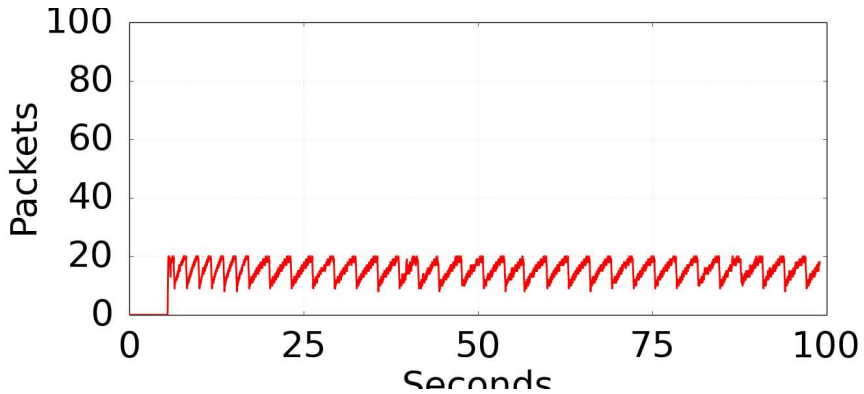
The CWND for TCP Reno will look like a sawtooth curve with an exponential slow start phase and once it reaches a saturation threshold, it will increase linearly until it reaches Wmax. Once it reaches Wmax, it will have a steep decrease. If network bandwidth per flow increases, the sawtooth will have some higher spikes than previous peaks corresponding to higher Wmax. If network bandwidth per flow decreases, the peaks will be lower than previous peaks

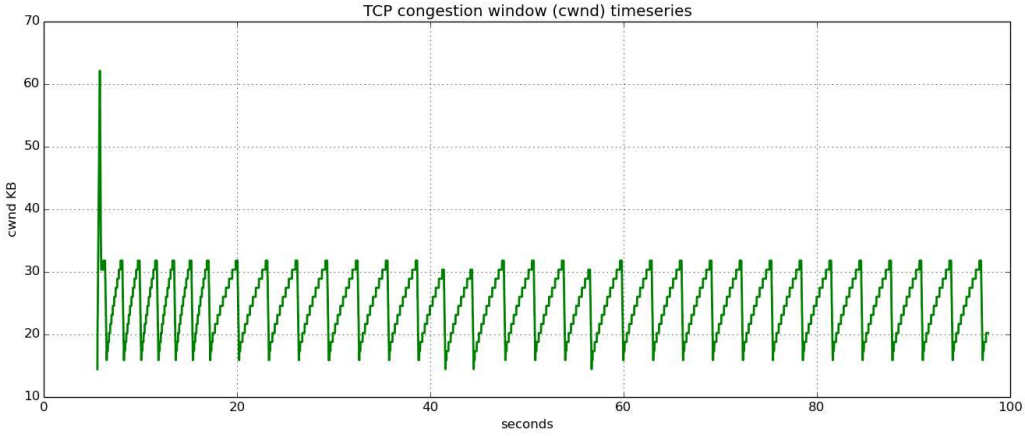
Part 1 TCP Reno

Item #3: With respect to queue occupancy, how are the graphs different? What do you think causes these differences to occur? (Make sure to include the graphs with your answer).

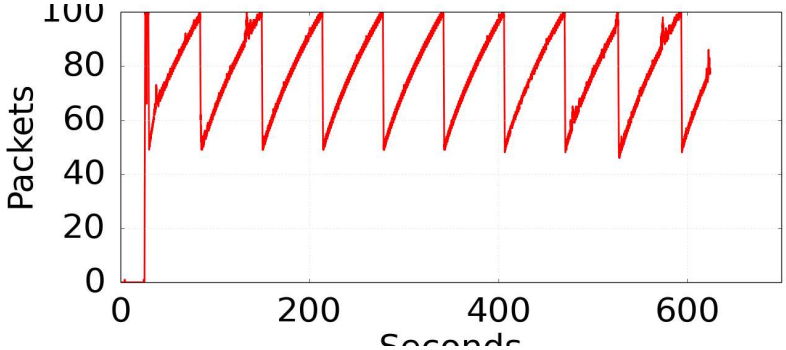
In the graph with small queue, we can see a higher frequency of congestion window growing and falling rapidly while in the graph with the large queue, we see that the congestion window grows at a slower rate. The rate at which queue is cleared is independent of the queue size and remains constant for our experiment setup. Therefore it will take a longer time for the packets to be cleared in the larger queue compared to the smaller queue in which less packets will be in queue.

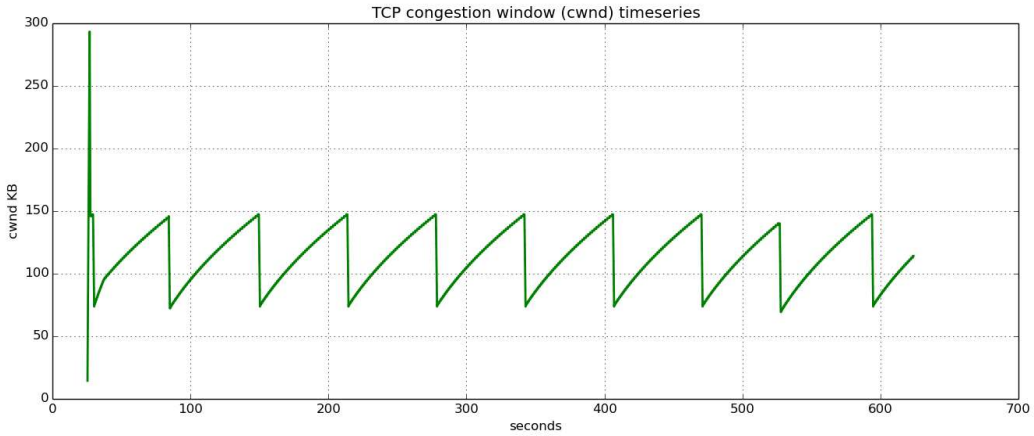
Small Queue





Large Queue





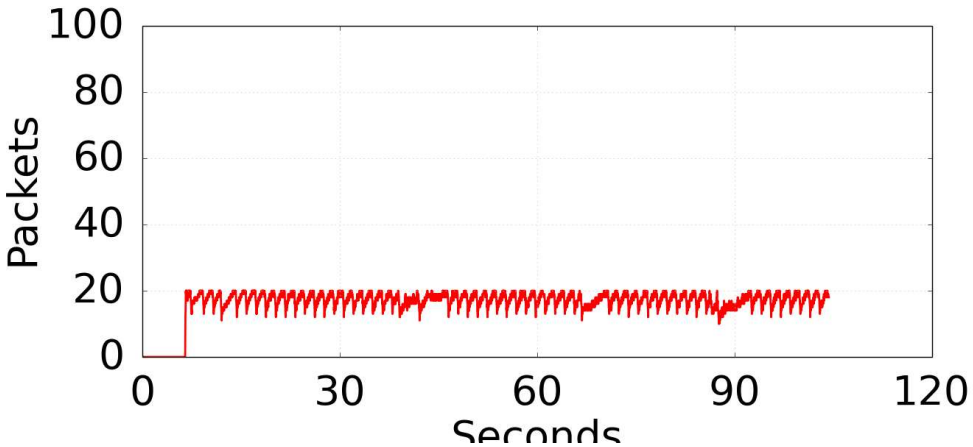
Part 2 TCP Cubic

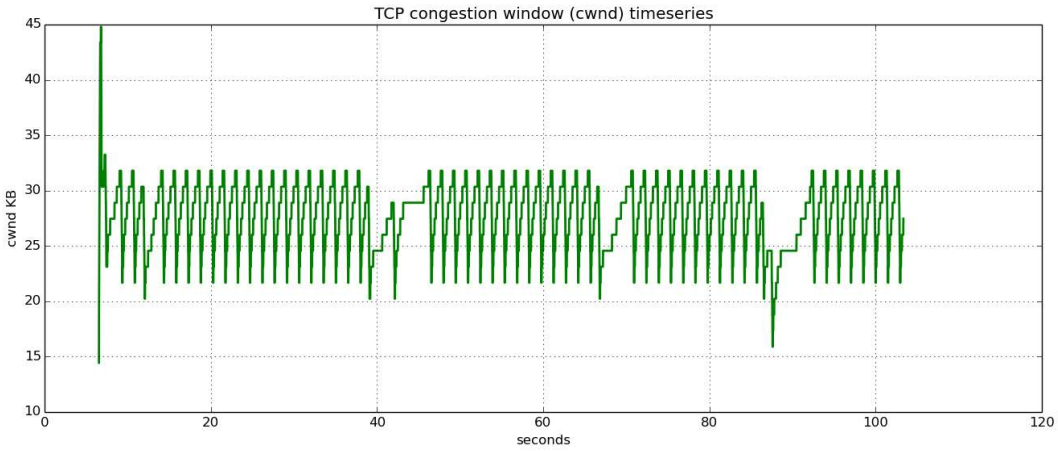
Item #4: What did you actually see in your CUBIC results? What anomalies or unexpected results did you see, and why do you think these behaviors/anomalies occurred? Be sure to reference the paper you read at the beginning of project the to help explain something that you saw in your results. Your hypothesis doesn't necessarily have to be correct, so long as it demonstrates understanding of the paper.

I expected cwnd to grow in a cubic shape since the beginning, exhibiting concave and convex regions from the start. However, it started growing linearly till the first packet loss occurred. This can be explained by the TCP friendliness of TCP Cubic. In this region, AIMD is in action which explains the linear growth of the cwnd curve initially.

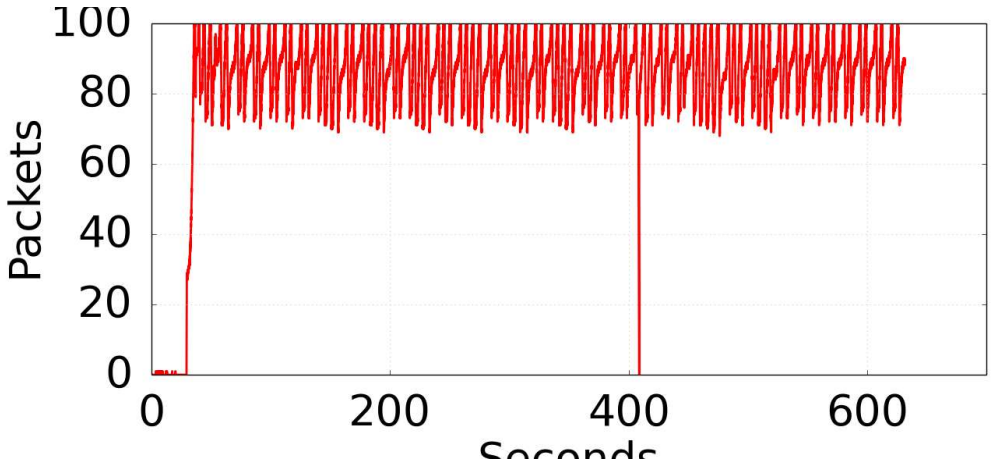
I also expected the cwnd curve to be in concave region only if bandwidth of the system remained constant, but for large queue I observed a different behaviour. The cwnd curve would first surpass Wmax and go into the convex region, the drop to a lower value on packet loss and again reach the peak, but this time remaining in the concave region only. This is explained by the Packet loss module in the algorithm.

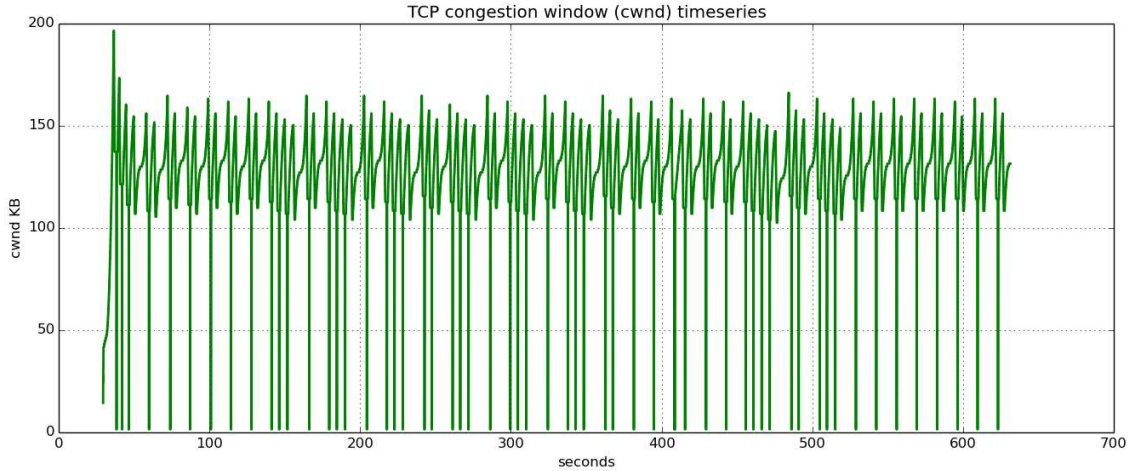
Small Queue




TCP congestion window (cwnd) timeseries
cwnd KB
seconds
45
40
35
30
25
20
15
10
0
20
40
60
80
100
120

Large Queue


Packets
100
80
60
40
20
0
0
200
400
600


TCP congestion window (cwnd) timeseries
cwnd KB
seconds
200
150
100
50
0
0
100
200
300
400
500
600
700

Item #5: Compare your prediction from Item #1 with the congestion window graphs you generated for TCP Reno and TCP CUBIC. How well did your predictions match up with the actual results? If they were different, what do you think caused the differences? Which queue size better matched your predictions, small or large?

For the most part, my predictions for both TCP Reno and TCP CUBIC were correct. The graphs for TCP Reno exhibited a sawtooth behavior while TCP CUBIC exhibited both concave and convex regions when bandwidth varied. The only discrepancy I saw was from the large queue TCP CUBIC. As mentioned, I expected the cwnd curve to be in concave region only if bandwidth of the system remained constant but the cwnd curve would first surpass Wmax and go into the convex region, the drop to a lower value on packet loss and again reach the peak, but this time remaining in the concave region only. Due to this discrepancy, the smaller queue sizes better matched my predictions for TCP Reno and TCP CUBIC.

Part 3 Resource Contention (short flow: downloading webpage vs long flow: streaming video)

Item #6: How does the presence of a long lived flow on the link affect the download time and network latency (RTT)? What factors do you think are at play causing this? Be sure to include the RTT and download times you recorded throughout Part 3.

The presence of long lived flow increased the download time and network latency. Before streaming the video, the initial download time of the webpage was 1 sec with average RTT to be 20.826 ms. However, after streaming the video, the download time of the webpage was 4.6s with average RTT to be 582.342. This is because when streaming the video, there are a lot of packets placed into the queue and when we request a download of the webpage, the packets from the short flow are added to the back of the queue. Since the bottleneck queue draining rate is constant, it must wait for iperf packets to be drained before draining the wget packets. Thus, it takes a longer time for download and RTT is much higher.

TCP RENO IPERF CONGESTION CONTROL ALGO

Initial download

time 100%[==========>] 177,669 175KB/s in 1.0s

rtt min/avg/max/mdev = 20.491/20.826/21.438/0.352 ms

After iperf

rtt min/avg/max/mdev = 557.582/582.342/615.248/17.931 ms

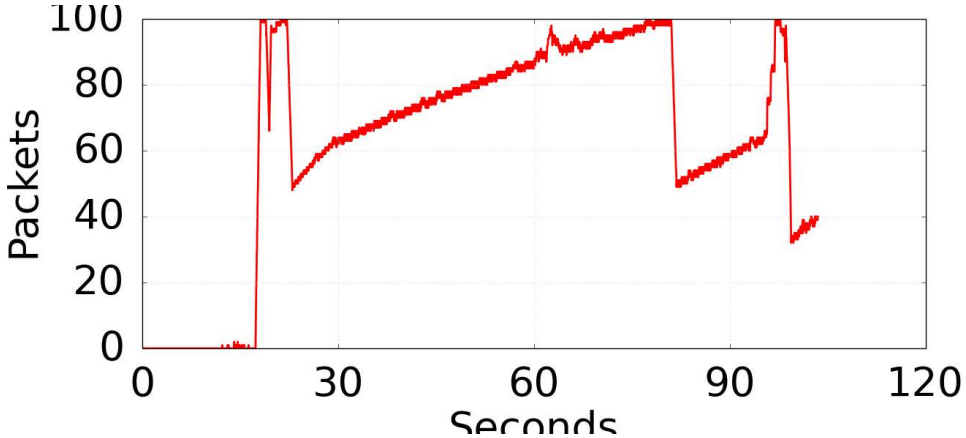
100%[==========>] 177,669 38.1KB/s in 4.6s

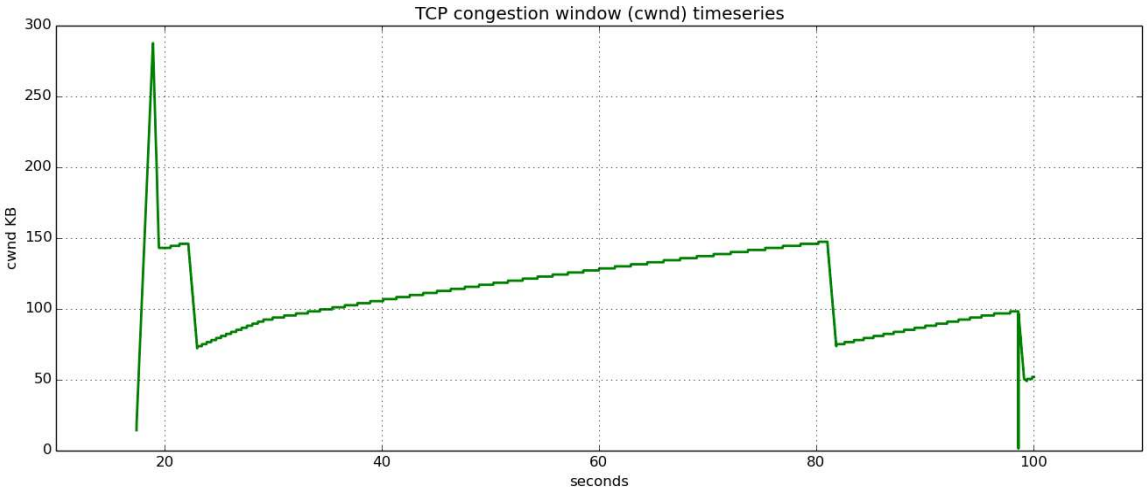
Part 4 Large vs Small Queues Performance

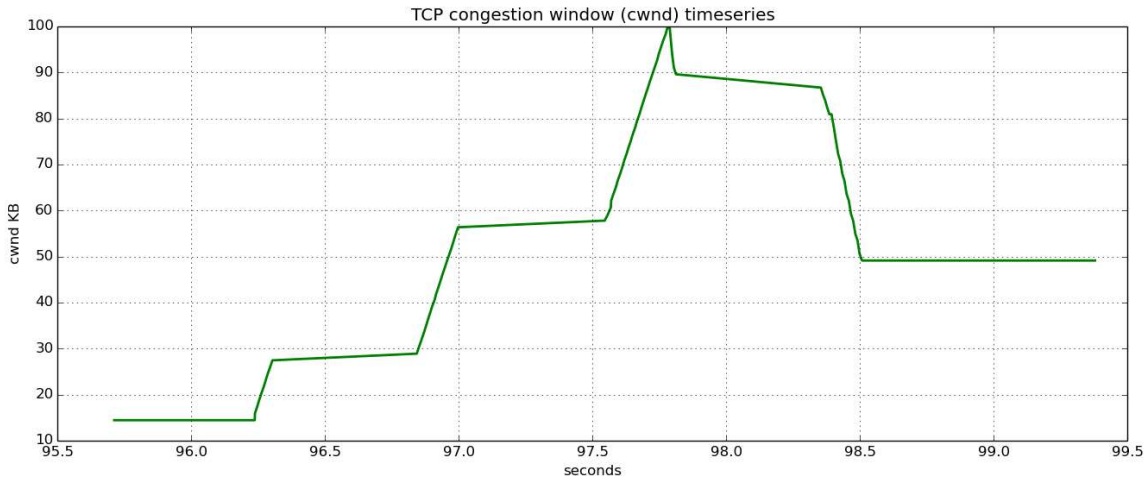
Item #7: How does the performance of the download differ with a smaller queue versus a larger queue? Why does reducing the queue size reduce the download time for wget? Be sure to include any graphs that support your reasoning.

The performance of smaller queue was much better than the performance of larger queue. In the larger queue graph for wget, we can see significant delays for wget as the queue is filled with iperf packets while in the small queue wget graph, we see a much shorter time for cwnd to increase. In addition, for the small queue iperf graph, it exhibits the expected sawtooth behavior while for the large queue graph, there seems to be some erratic behavior which explains a possible cause for buffer bloat. The download time for wget was much less for smaller queues than larger queues. This is because the rate at which queue is cleared is independent of the queue size and remains constant. So when few packets of the webpage are queued at the end of larger queue, it will take long time for the previous packets to be cleared, compared to the smaller queue in which less packets will be in queue ahead of the webpage packets, and hence those lesser number of packets would require less time to get cleared even at the same rate.

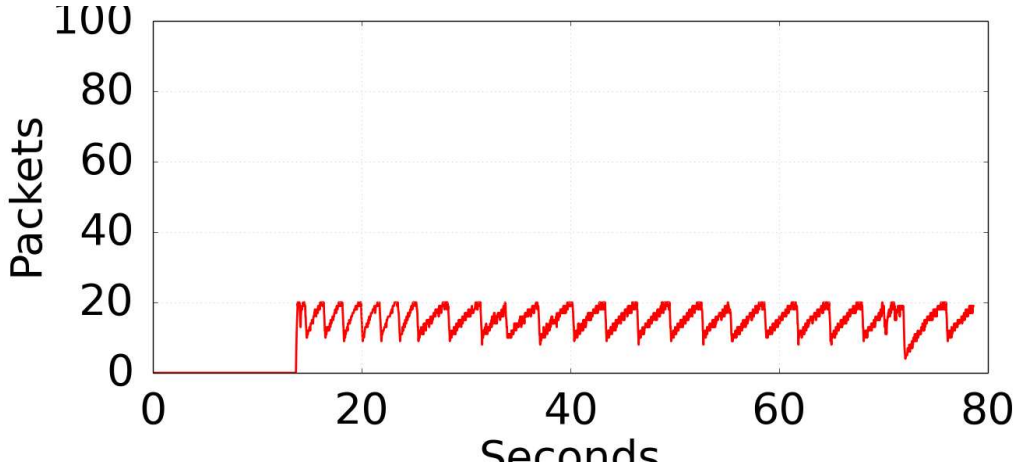
Large Queue

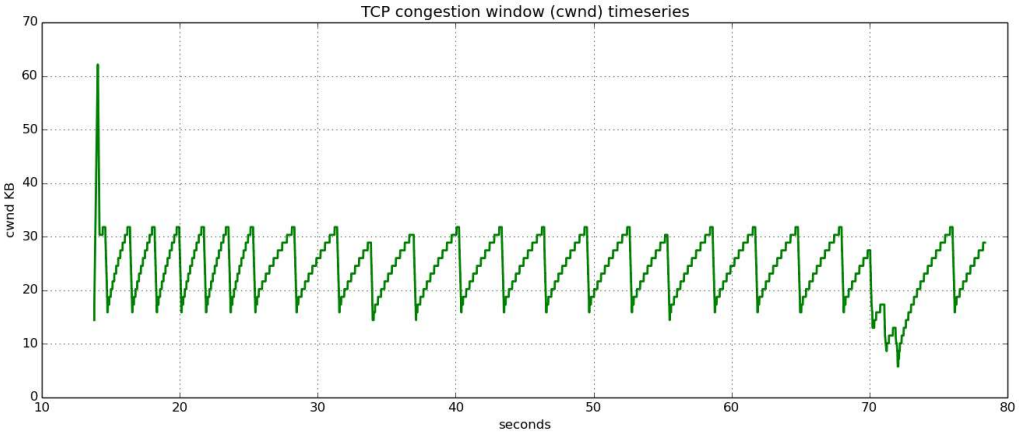


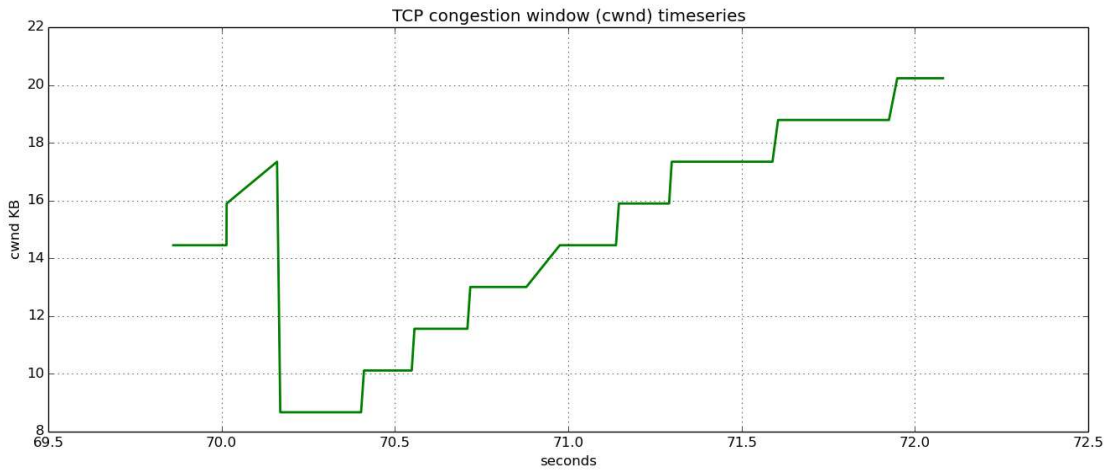



TCP congestion window (cwnd) timeseries
cwnd KB
seconds
100
90
80
70
60
50
40
30
20
10
95.5
96.0
96.5
97.0
97.5
98.0
98.5
99.0
99.5

Small Queue


Packets
Seconds
100
80
60
40
20
0
0
20
40
60
80


TCP congestion window (cwnd) timeseries
cwnd KB
seconds
70
60
50
40
30
20
10
0
10
20
30
40
50
60
70
80



Part 5: Using Traffic Control to Reduce Buffer Bloat

Item #8: How does the presence of a long lived flow on the link affect the download time and network latency (RTT) when using two queues? Were the results as you expected? Be sure to include the RTT and download times you recorded throughout Part 5.

Using two queues, we can see dramatic improvement of the download time and RTT. Before the presence of long flow, the average RTT is 21.478 ms and download time for webpage is 1 second. After the presence of long flow, the average RTT is 21.085 sec and download time for webpage is 2 seconds. The results were as expected because by using two queues with a shared bottle neck drain rate, packets from iperf as well as wget can be processed in parallel.

Before Long-lived flow

rtt min/avg/max/mdev = 20.180/21.478/24.332/1.262 ms

100%[==>] 177,669 175KB/s in 1.0s

After Long-lived flow

rtt min/avg/max/mdev = 20.173/21.085/24.030/1.100 ms

100%[==>] 177,669 87.3KB/s in 2.0s

Part 6 Repeat Part 3-4 for TCB Cubic

Item #9: Did the change in congestion control algorithm result in any significant differences in the results? ● If so, which algorithm performed better? What aspect of that particular algorithm do you think caused it to perform better? ● If there was no change, was this what you expected? Why or why not?

TCP CUBIC algorithm resulted in a better performance as shown in the graphs below. For large queues, we can specifically see the improvement. Recall the iperf and wget graphs of large queues for TCP Reno. It did not exhibit any sawtooth behavior and we can see the huge delays of wget caused by the large amount of iperf packets. Contrast the graph with TCP CUBIC, which exhibits the concave/convex regions and wget graph exhibit less delays. TCP CUBIC performed better because it simplifies the BIC-TCP window control by providing a cubic function and improves its TCP-friendliness and RTT-fairness. The key feature of CUBIC is that its window growth depends only on the time between two consecutive congestion events. Thus, the window growth becomes independent of RTTs. This feature allows CUBIC flows competing in the same bottleneck to have approximately the same window size independent of their RTTs, achieving good RTT-fairness. Furthermore, when RTTs are short, since the window growth rate is fixed, its growth rate could be slower than TCP standards. Since TCP standards work well under short RTTs, this feature enhances the TCP-friendliness of the protocol.

TCP CUBIC IPERF CONGESTION CONTROL ALGO

Before iperf

100%[==>] 177,669 175KB/s in 1.0s

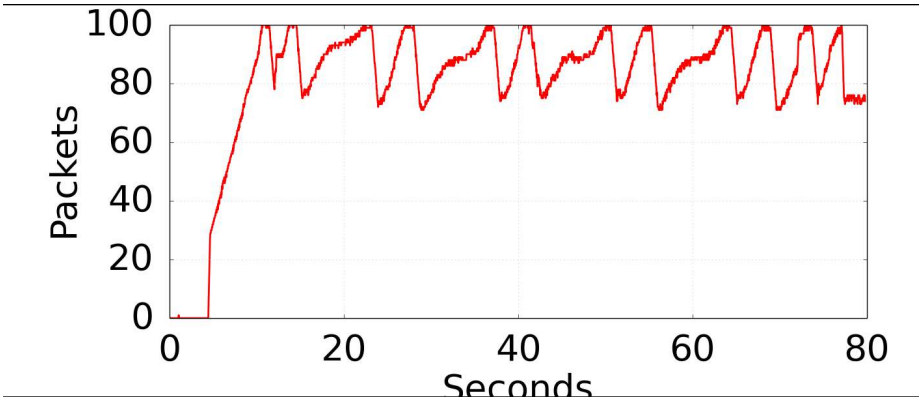
rtt min/avg/max/mdev = 20.206/20.970/23.912/1.063 ms

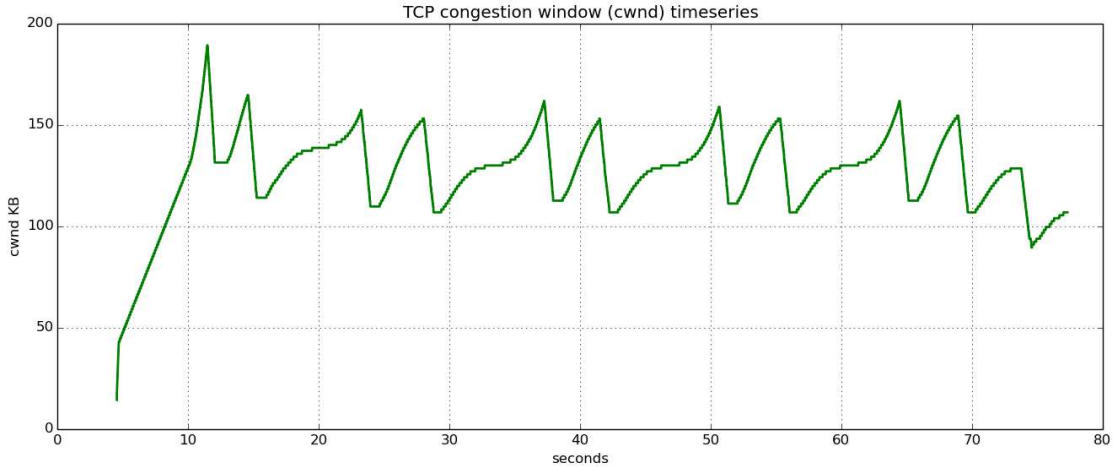
after iperf to simulate long flows(AIMD)

rtt min/avg/max/mdev = 588.347/697.579/810.256/79.571 ms

100%[==>] 177,669 39.3KB/s in 4s

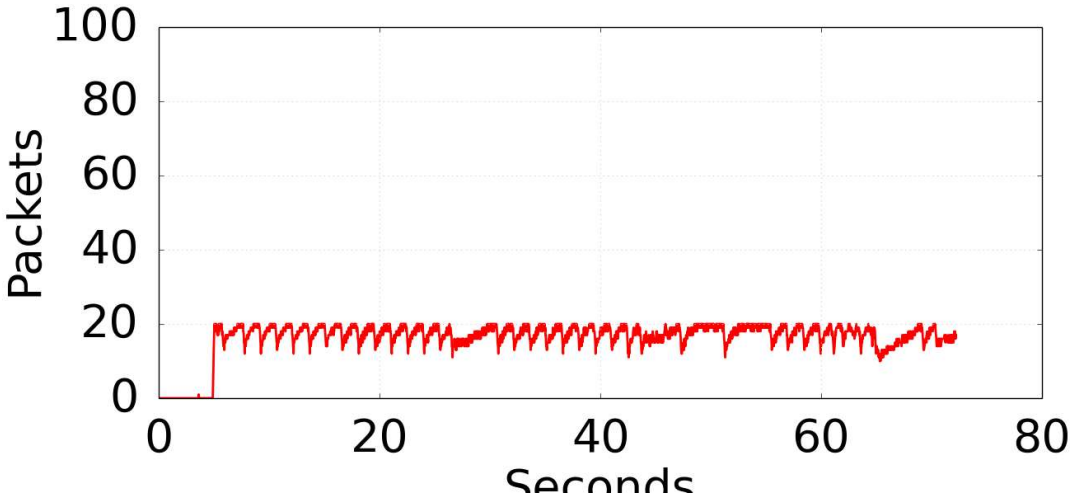
Large Queue

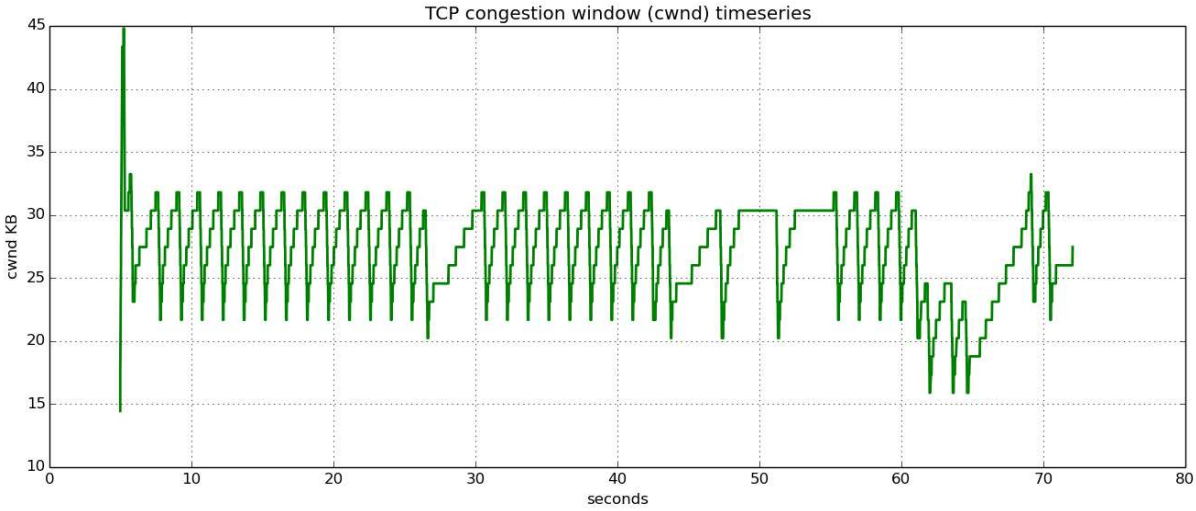


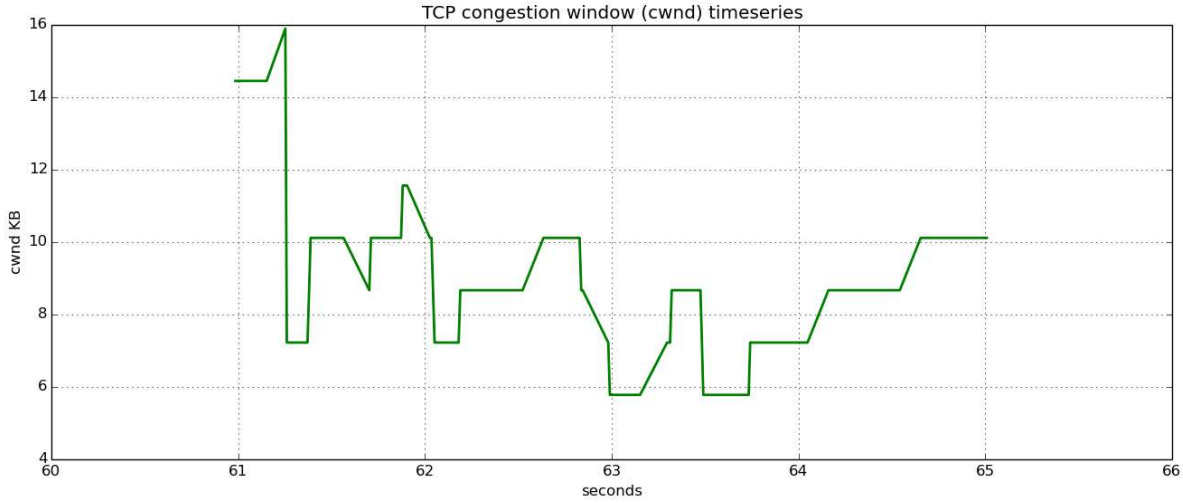

TCP congestion window (cwnd) timeseries
cwnd KB
seconds
0
50
100
150
200
0
10
20
30
40
50
60
70
80


TCP congestion window (cwnd) timeseries
cwnd KB
seconds
10
15
20
25
30
35
40
72
73
74
75
76
77
78

Small Queue


Packets
Seconds
0
20
40
60
80
100
0
20
40
60
80


TCP congestion window (cwnd) timeseries
cwnd KB
seconds
45
40
35
30
25
20
15
10
0
10
20
30
40
50
60
70
80


TCP congestion window (cwnd) timeseries
cwnd KB
seconds
16
14
12
10
8
6
4
60
61
62
63
64
65
66