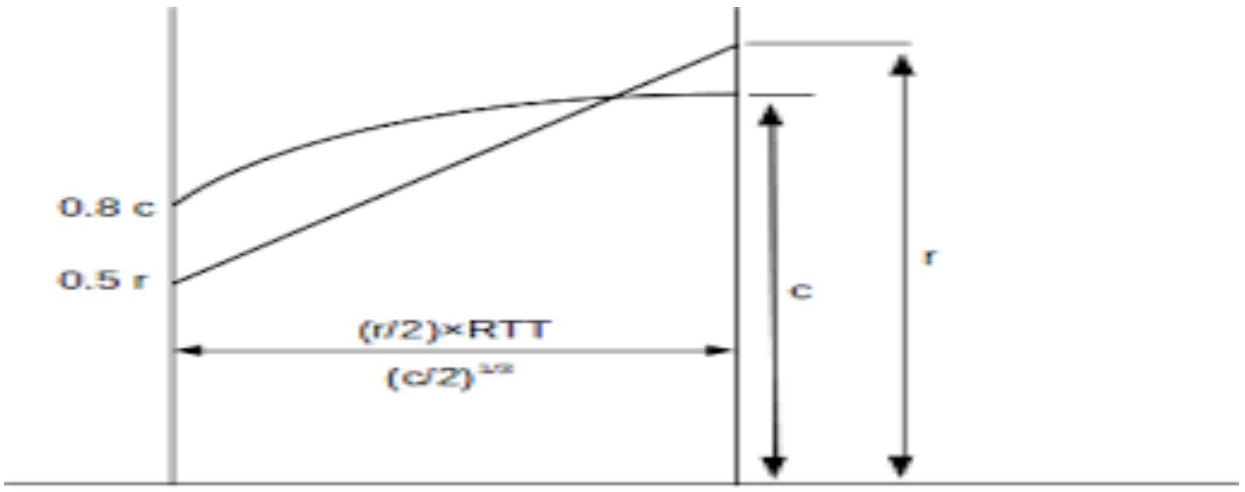
Name: Imran Yousuf
Project 6: Congestion Control & Buffer Bloat

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.

From the paper and my understanding the TCP cubic will be flattened after a long live TCP flow while TCP Reno will be linearly increasing.



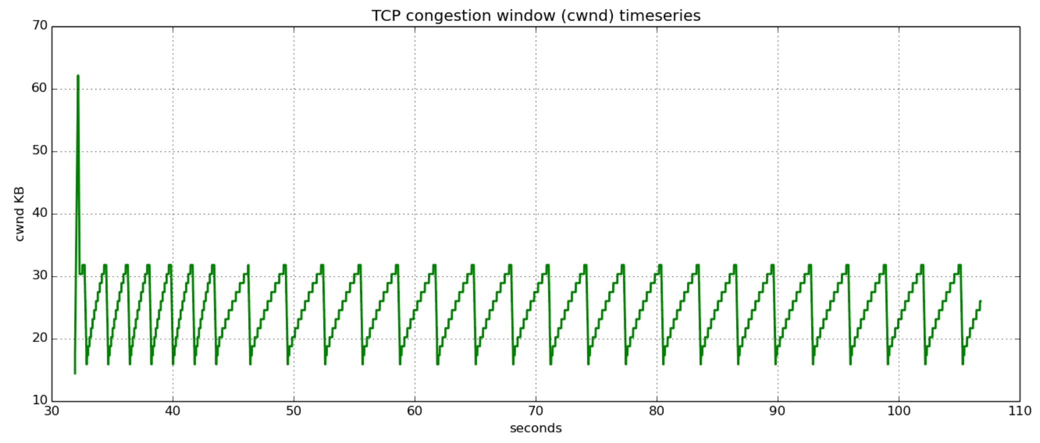
One tooth, TCP Cubic v TCP Reno

Item #2: Explain why you expect to see this; specifically relate your answers to details in the paper.

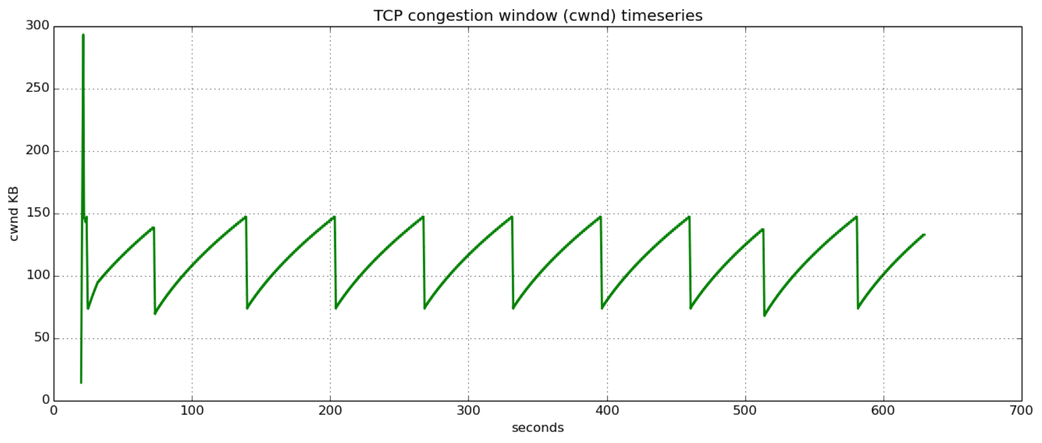
The reason for longer live TCP cubic is faster because when TCP RENO grows its window one per round trip time (RTT). It makes the data transport speed of TCP used in all major operating systems slow, thus extremely under-utilizing the networks especially if the length of flows is much shorter than the time TCP grows its windows to the full size of the BDP of a path. So basically, if a flow finishes before that time, it severely under-utilizes the path. But in CUBIC the window growth depends only on the real time between two consecutive congestion events. One congestion event is the time when TCP undergoes fast recovery. So it does not under-utilize the path. Thus, the window growth becomes independent of RTTs. This allows CUBIC flows competing in the same bottleneck to have approximately the same window size independent of their RTTs, achieving good RTT-fairness, making the flow finish faster in the long run.

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).

Small Queue:



Large Queue:



The larger queue had a higher queue occupancy. It almost had a 5 times more occupancy. I think the reason is because the RTT is longer.

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.

For shorter queue the CUBIC results were slow and for larger queue CUBIC results were faster. The reason is because when RTTs are short, since the window growth rate is fixed, its growth rate could be slower than standards as it does work well under short RTTs, this feature enhances the TCP-friendliness of the protocol. The reason for longer queue it was faster because TCP undergoes fast recovery.

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?

My predictions were very close, they matched up nicely. The large queue matched a little better than the small queue.

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.

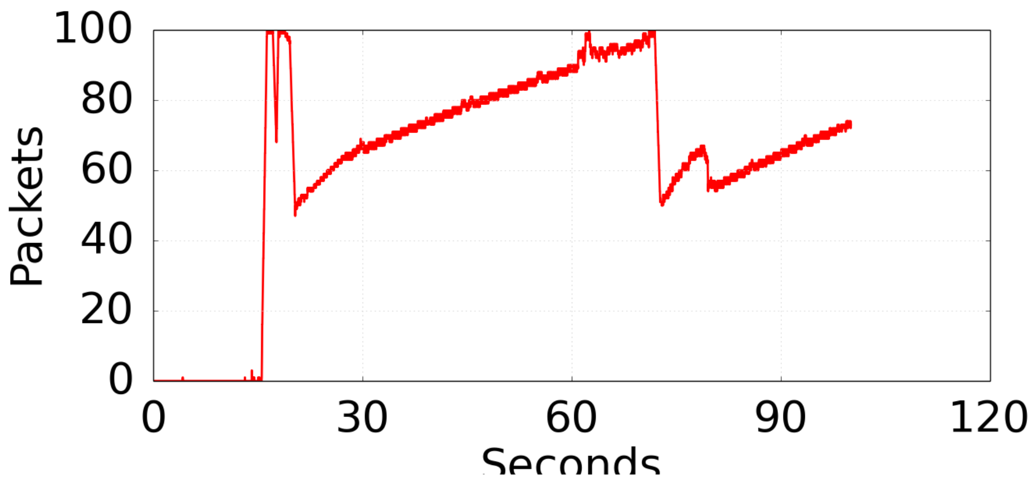
It slows it down because it doesn't work so well for smaller data transfers, because in order to make sure it doesn't overwhelm the network, TCP uses a "slow start" mechanism. For a long download, the slow start portion is only a fraction of the total time, but for short transfers, by the time TCP gets up to speed, the transfer is already over. Because TCP has to wait for acknowledgments from the receiver, more latency means more time spent in slow start.

```
mininet> h2 tail -f ./iperf-recv.txt
[  4]  7.0- 8.0 sec   173 KBytes  1.42 Mbits/sec
[  4]  8.0- 9.0 sec   173 KBytes  1.42 Mbits/sec
[  4]  9.0-10.0 sec   173 KBytes  1.42 Mbits/sec
[  4] 10.0-11.0 sec   173 KBytes  1.42 Mbits/sec
```

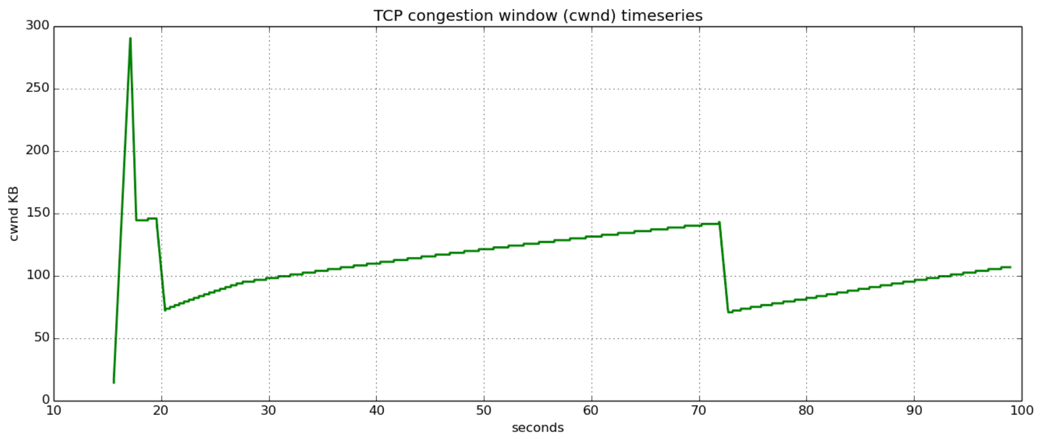
```
mininet>  h2 wget http://10.0.0.1
--2017-10-30 04:06:13--  http://10.0.0.1/
Connecting to 10.0.0.1:80... connected.
HTTP request sent, awaiting response... 200 OK
Length: 177669 (174K) [text/html]
Saving to: 'index.html'

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

2017-10-30 04:06:14 (175 KB/s) - 'index.html' saved [177669/177669]

--- 10.0.0.2 ping statistics ---
10 packets transmitted, 10 received, 0% packet loss, time 9010ms
rtt min/avg/max/mdev = 20.110/20.637/22.057/0.718 ms
```

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.

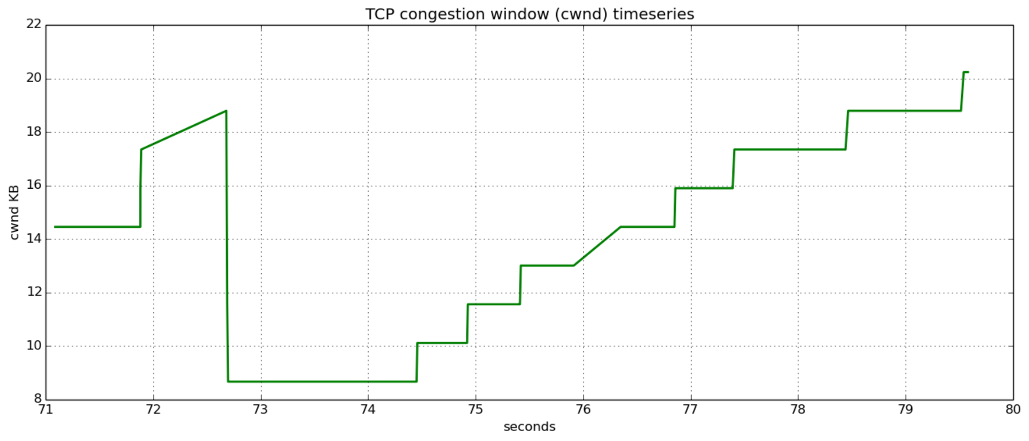
Smaller Queue



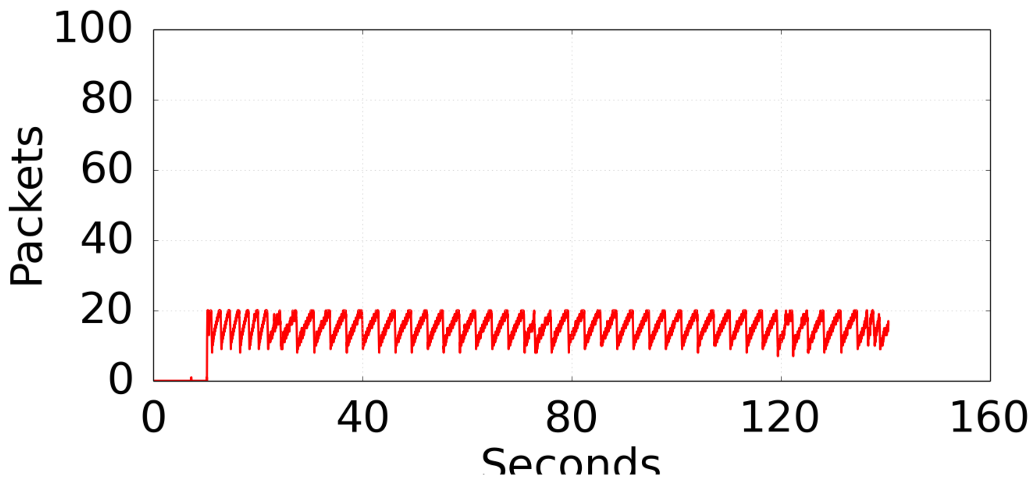
Smaller Queue iperf



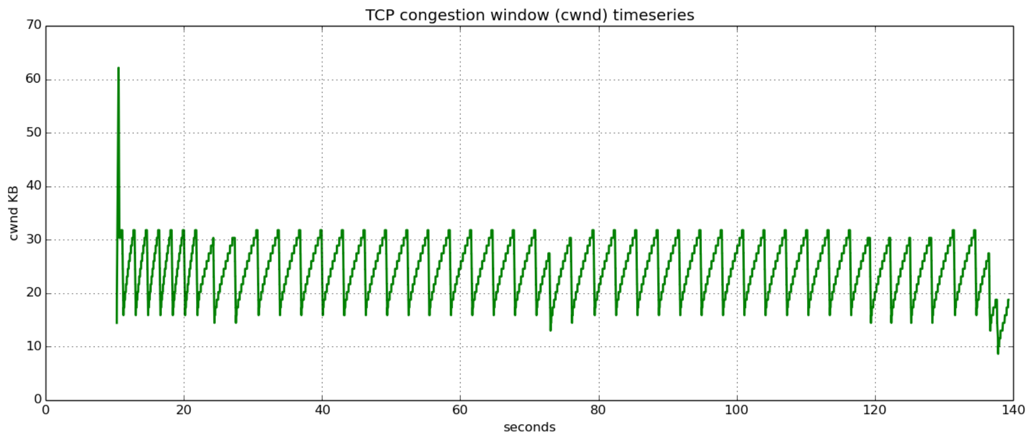
Smaller Queue wget



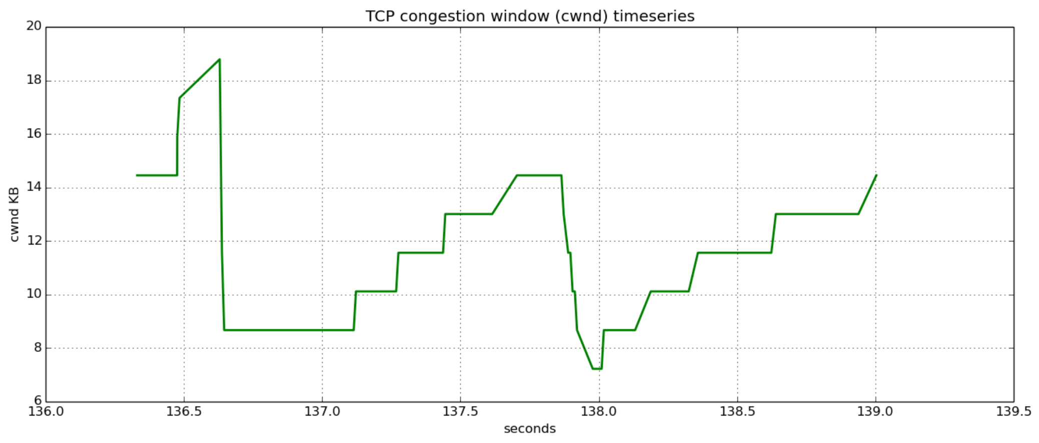
Larger Queue



Larger Queue iperf



Larger Queue wget



Download was faster in smaller queue than the larger queue unexpectedly. Reducing the queue size reduce the download time for wget because the window growth function of HTCP is a quadratic function of HTCP is unique in that it adjusts the decrease factor by a function of RTTs which is engineered to estimate the queue size in the network path of the current flow. Thus, the decrease factor is adjusted to be proportional to the queue size.

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.

```
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=20.3 ms
64 bytes from 10.0.0.2: icmp_seq=2 ttl=64 time=20.1 ms
64 bytes from 10.0.0.2: icmp_seq=3 ttl=64 time=20.1 ms
64 bytes from 10.0.0.2: icmp_seq=4 ttl=64 time=20.1 ms
64 bytes from 10.0.0.2: icmp_seq=5 ttl=64 time=20.1 ms
64 bytes from 10.0.0.2: icmp_seq=6 ttl=64 time=20.1 ms
64 bytes from 10.0.0.2: icmp_seq=7 ttl=64 time=20.1 ms
64 bytes from 10.0.0.2: icmp_seq=8 ttl=64 time=20.1 ms
64 bytes from 10.0.0.2: icmp_seq=9 ttl=64 time=20.4 ms
64 bytes from 10.0.0.2: icmp_seq=10 ttl=64 time=20.2 ms

--- 10.0.0.2 ping statistics ---
10 packets transmitted, 10 received, 0% packet loss, time 9009ms
rtt min/avg/max/mdev = 20.104/20.199/20.424/0.133 ms
```

```
mininet> h2 wget http://10.0.0.1
--2017-10-30 01:57:18--  http://10.0.0.1/
Connecting to 10.0.0.1:80... connected.
HTTP request sent, awaiting response... 200 OK
Length: 177669 (174K) [text/html]
Saving to: ‘index.html’

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

2017-10-30 01:57:19 (175 KB/s) - ‘index.html’ saved [177669/177669]
```

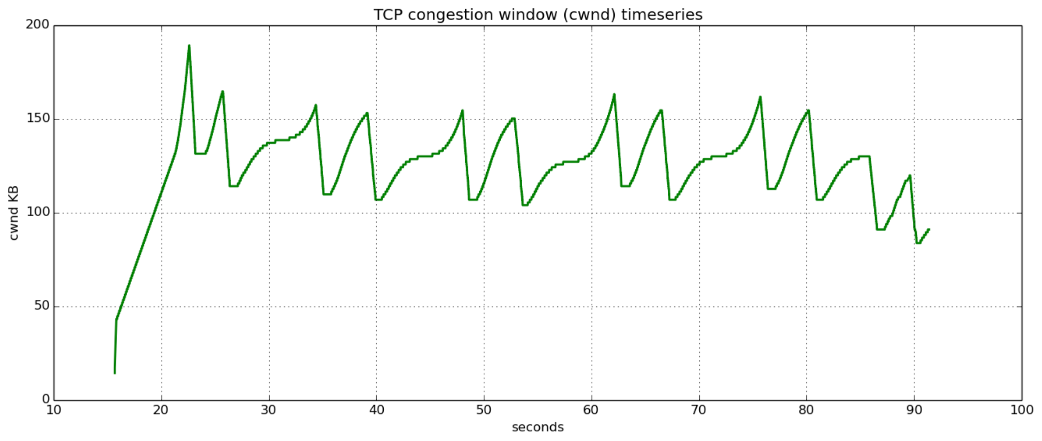
Longer --→

```
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=21.1 ms
64 bytes from 10.0.0.2: icmp_seq=2 ttl=64 time=21.3 ms
64 bytes from 10.0.0.2: icmp_seq=3 ttl=64 time=20.3 ms
64 bytes from 10.0.0.2: icmp_seq=4 ttl=64 time=20.2 ms
64 bytes from 10.0.0.2: icmp_seq=5 ttl=64 time=20.1 ms
64 bytes from 10.0.0.2: icmp_seq=6 ttl=64 time=20.1 ms
64 bytes from 10.0.0.2: icmp_seq=7 ttl=64 time=20.8 ms
64 bytes from 10.0.0.2: icmp_seq=8 ttl=64 time=20.1 ms
64 bytes from 10.0.0.2: icmp_seq=9 ttl=64 time=20.0 ms
64 bytes from 10.0.0.2: icmp_seq=10 ttl=64 time=20.1 ms

--- 10.0.0.2 ping statistics ---
10 packets transmitted, 10 received, 0% packet loss, time 9010ms
rtt min/avg/max/mdev = 20.082/20.444/21.336/0.494 ms


mininet> h2 wget http://10.0.0.1
--2017-10-30 02:08:04--  http://10.0.0.1/
Connecting to 10.0.0.1:80... connected.
HTTP request sent, awaiting response... 200 OK
Length: 177669 (174K) [text/html]
Saving to: ‘index.html.1’

100%[======================================>] 177,669    87.4KB/s   in 2.0s

2017-10-30 02:08:06 (87.4 KB/s) - ‘index.html.1’ saved [177669/177669]
```

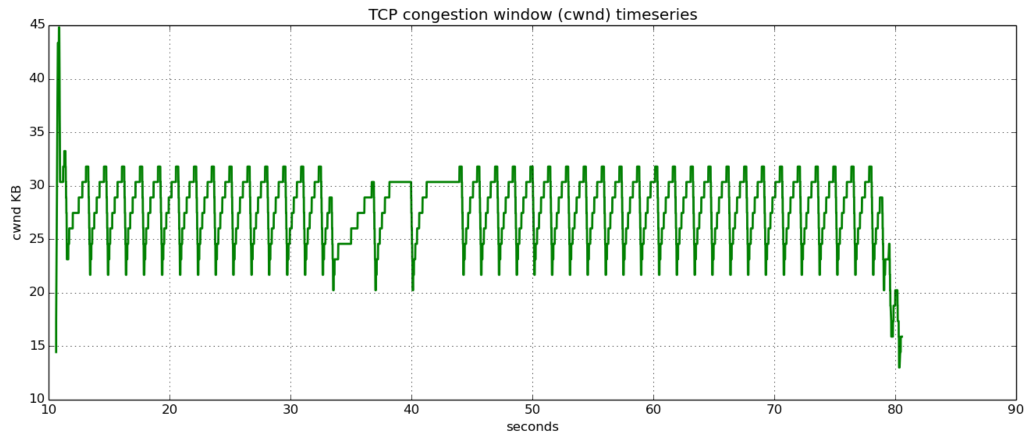
*The results were expected. It takes longer in presence of a long lived flow on the link affect the download time and network latency when using two queues.

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?

Experiment -3



Experiment - 4



The change in congestion control algorithm resulted in an improvement in the results. TCP CUBE performed better in the long live TCP flow and TCP Reno performed better in shorter ones. The window growth rate made TCP cube perform better. I expected more significant change.