Project 5 by Neeraj Sharma

Item#1: Curve expectation

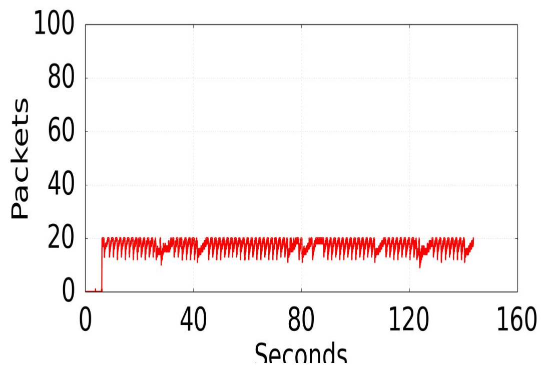
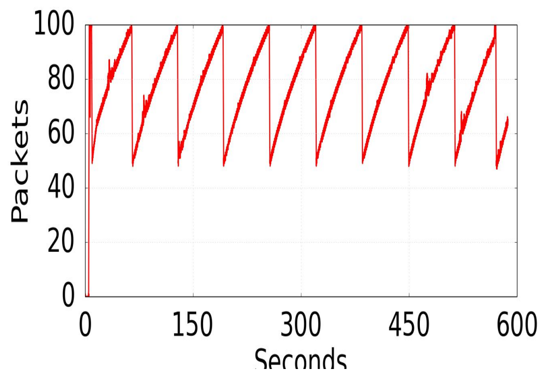
For a long-lived TCP flow using TCP CUBIC, we expect the CWND curve to grow in a concave profile initially until it reaches Wmax. After that, the pattern of growth turns into a convex profile. For a long-lived TCP flow using TCP RENO, we should expect to see classic TCP Sawtooth pattern.

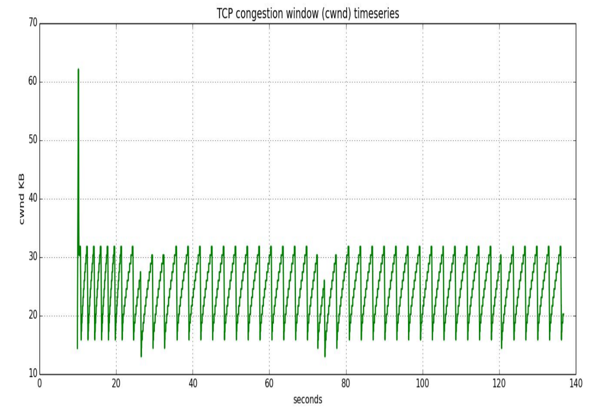
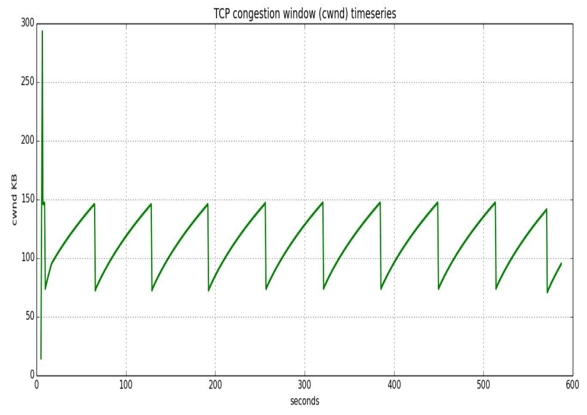
Item#2: Prediction justification

Concave and convex profile for CUBIC happens because the window growth function is a cubic function. Initially, when the CWND is less than the window size standard TCP would reach, CWND will be in TCP Mode. If it's not in TCP mode, if CWND < Wmax, it's in concave region and if CWND > Wmax, it's in the convex region.
Sawtooth pattern for TCP Reno happens as it grows initially to a peak, called the "slow-start" phase. It'll grow to a point where it start seeing packet loss and enters fast recovery and congestion avoidance. At this time, following the AIMD pattern, it drops the sending rate and slowly adds to the rate until it starts seeing packet loss again. This generates a sawtooth pattern.

Item#3: Difference in TCP Reno iperf graphs w.r.t. Queue occupancy

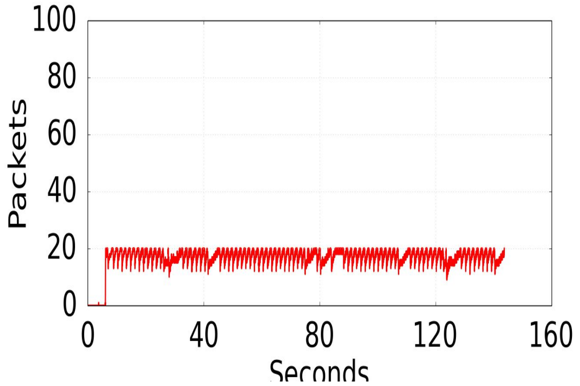
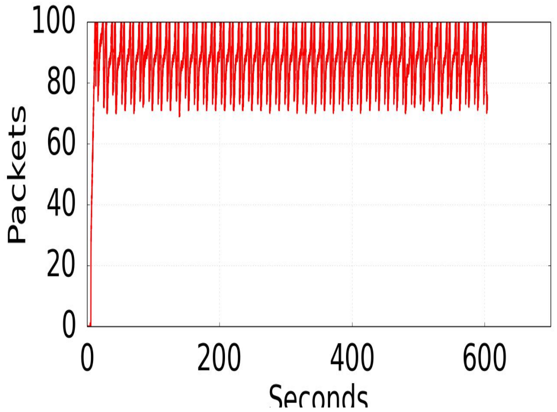
With the small queue, there are at max 20 packets sent on the network and congestion window size is smaller. Because of the small size, it reaches it's Wmax quickly and drops to avoid congestion. Thus, small queue observes high frequency of TCP sawtooth, as the window grows quickly and falls quickly. With the large queue, there are at max 100 packets sent on the network and congestion window is larger. It takes longer to reach it's Wmax with additive increase in this case. The rate at which the sender treats the message from the queue is the same and therefore, the larger queue takes longer to treat the packets, as there are more packets.

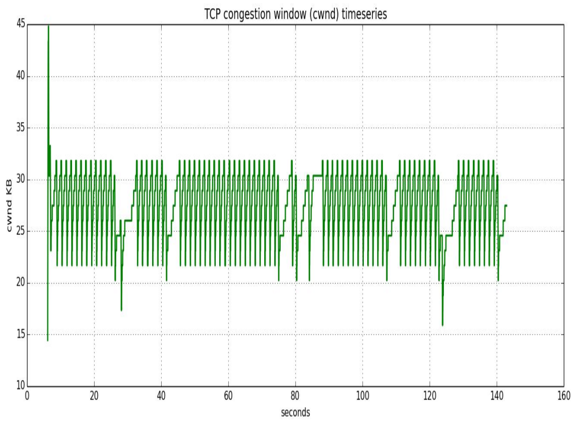
Small Queue	Large Queue
	

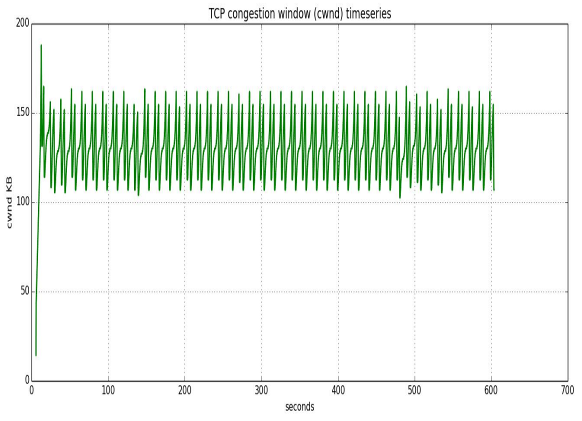
	

Item#4: TCP Cubic Graphs

In both small and large queue cases, the CWND increases linearly first as it exhibits the TCP slow-start phase. This is expected because CUBIC remains in the TCP friendly region initially. As it sees a packet-loss, it starts the concave growth profile and after the it reaches Wmax, it grows with the convex profile. The graph seems a bit spikey in its condensed mode but when I zoomed in and extrapolated the pattern, it demonstrates the expected behavior. One unexpected behavior I see is in case of small queue, CWND grows in a concave profile and convex profile sometimes, but majority of times, it exhibits rather TCP like behavior. In case of large queue however, the growth rate is as expected, it starts with a slow-start and follows the concave growth and convex growth before dropping to start concave growth again. This could be because of the sampling rate for collecting this data.

Small Queue Cubic	Large Queue Cubic
	





Item#5: Comparing prediction

For TCP Reno, the graphs matched as I predicted. The one with the large queue matches better, as with the small queue, there are some anomalies in the steady-state behavior where TCP drops CWND with a smaller Wmax.

For TCP CUBIC, the graphs matched as I predicted. The one with the large queue matches better. The smaller queue might have lower sampling rate or with the same sampling rate, less data points as the queue size was small. That's why, the graph isn't exactly concave and convex overall. The one with the large queue matches the growth profile of cubic function with the slow-start TCP friendly region.

Item#6: Affect on RTT and Latency by a long-lived flow

The presence of a long-lived flow increased the round-trip time and the latency of http requests. As you can see from the data, RTT increased from 20ms to 690ms and download time increased from 1 second to 7 seconds. This happens because the queue already contains packets from long-lived flow when the short-lived flow packets arrive. As the sender drains the queue at the same rate, the short-lived flow takes longer to serve as the packets are pushed to the queue and not sent immediately.

Before	After

`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=21.2 ms`
`64 bytes from 10.0.0.2: icmp_seq=3 ttl=64 time=20.2 ms`
	`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=688 ms`
	`64 bytes from 10.0.0.2: icmp_seq=2 ttl=64 time=695 ms`
	`64 bytes from 10.0.0.2: icmp_seq=3 ttl=64 time=696 ms`

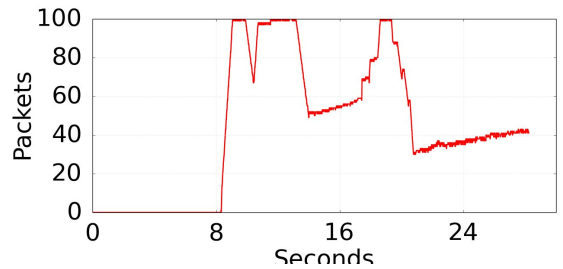
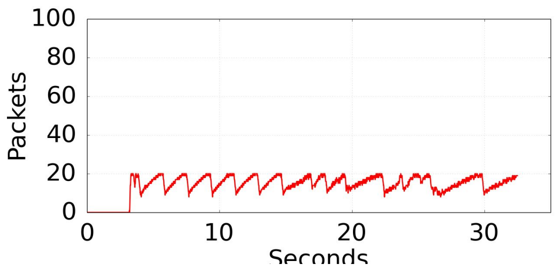
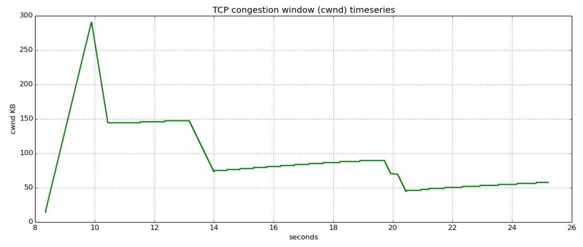
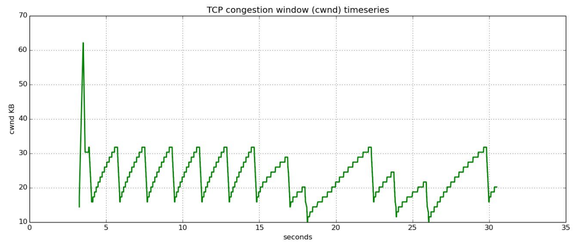
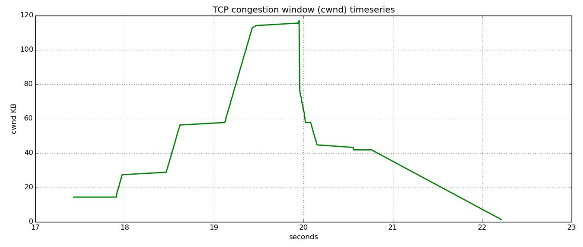
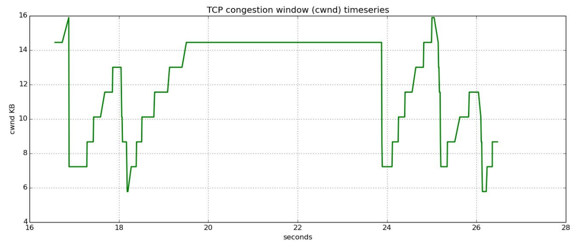
`mininet> h2 wget http://10.0.0.1`
`--2018-10-29 00:22:32--  http://10.0.0.1/`
	`mininet> h2 wget http://10.0.0.1`
	`--2018-10-29 00:25:14--  http://10.0.0.1/`

```
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
```

```
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     24.5KB/s   in 7.1s
```

Item#7: Improving resource contention

The download time improves with a small queue. This is a good example of a bufferbloat problem and resolution. In case of large queue, the iperf packets filled up the queue and because the queue draining rate is constant by the sender, it took longer to send the packets of wget requests as the packets were added to the back of the queue. With a smaller queue, the queue is drained faster. So, even though iperf was in progress and packets were continuously added to the queue and the packets of wget request were added to the back of the queue, the packets were sent quickly, as the queue size was 1/5th of the large queue size and it takes less time to treat messages from the queue. As it can be observed from the graphs, with a large queue in experiment 1, CWND has a very irregular pattern as well as the packets treatment. However, with a small queue, CWND exhibits the normal sawtooth pattern.

Experiment 1	Experiment 2
	
	
	

Item#8: Improving resource contention with two queues

Presence of a long-lived flow did increase the round-trip time and the download time but not as significantly as it did earlier with a large queue. The results were expected. I was expecting a slight increase in RTT and download time because two queues existed and scheduler implements fair queueing. Because both the queues have packets in it, the RTT and page load time will be increased a little bit as with fair queueing, both clients will get a fair share of the bandwidth. However, because we used two queues, the iperf packets did not overwhelm the buffer and the wget packets were not starved of the bandwidth. Hence, the increase in RTT and download time was smaller. As seen from data, RTT was slightly higher or even the same, but page-load time was increased by 1 second.

Without presence of long-lived flow	With presence of long-lived flow

`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=20.2 ms`
`64 bytes from 10.0.0.2: icmp_seq=3 ttl=64 time=20.9 ms`
	`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=20.4 ms`
	`64 bytes from 10.0.0.2: icmp_seq=2 ttl=64 time=21.8 ms`
	`64 bytes from 10.0.0.2: icmp_seq=3 ttl=64 time=21.0 ms`

`mininet> h2 wget http://10.0.0.1`
`--2018-10-29 00:56:04--  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’`
 	
`100%[=====================================>] 177,669      175KB/s   in 1.0s`
	`mininet> h2 wget http://10.0.0.1`
	`--2018-10-29 00:56:30--  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     87.3KB/s   in 2.0s`

Item#9: Resource contention and resolution observation with TCP CUBIC

The presence of a long-lived flow increased the round-trip time and the latency of http requests. As you can see from the data, RTT increased from 20ms to 600ms and download time increased from 1 second to 5 seconds. This is definitely an improvement over TCP RENO. CUBIC performs better than RENO for the same experiment and yields better results. There was not a huge improvement in round trip time, but wget request observed an improvement of 2.2 seconds compared to RENO. The CUBIC function allows for RTT independent growth which helps in the improved performance. CUBIC equation depends on t, K and Wmax and is independent of RTT. Experiment 3 still observes starvation because of large queue and smaller queue observes even better performance for the reasons already discussed. However, the concave growth profile and convex growth profile are evident in both cases and therefore, it exhibits a superior performance over RENO.

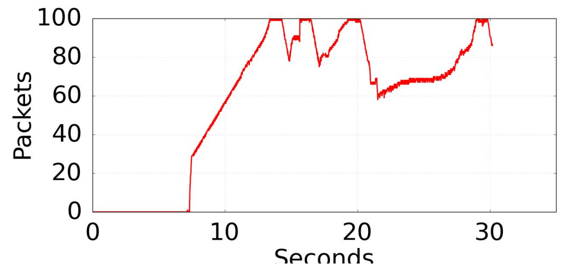
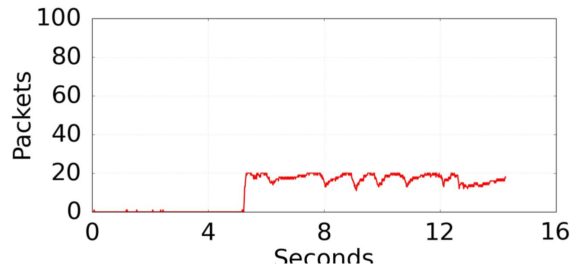
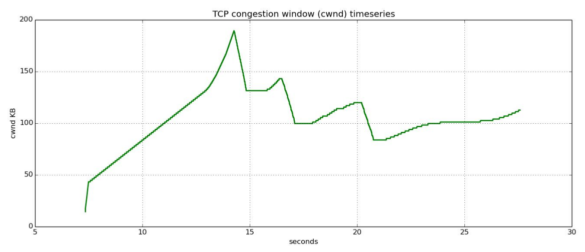
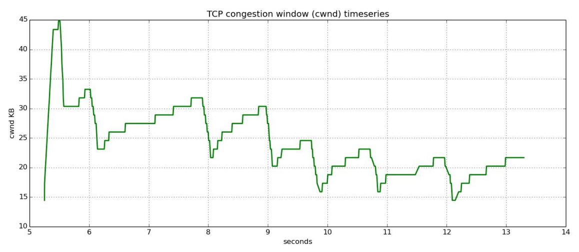
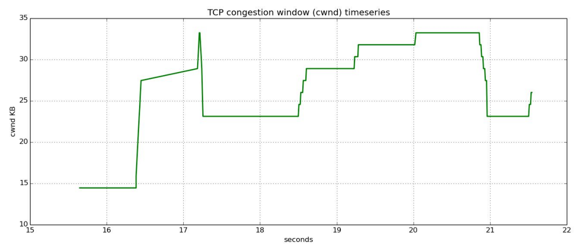
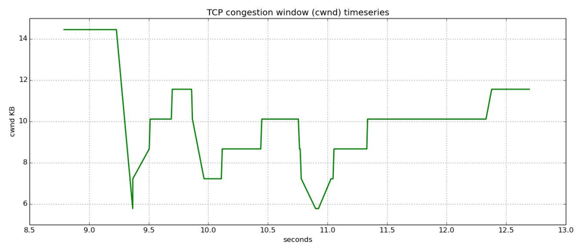
Before	After
`mininet> h1 ping -c 10 h2`	`mininet> h1 ping -c 10 h2`

```
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=20.4 ms
64 bytes from 10.0.0.2: icmp_seq=2 ttl=64
time=22.0 ms
64 bytes from 10.0.0.2: icmp_seq=3 ttl=64
time=20.2 ms
```

```
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=640 ms
64 bytes from 10.0.0.2: icmp_seq=2 ttl=64
time=580 ms
64 bytes from 10.0.0.2: icmp_seq=3 ttl=64
time=630 ms
```

```
mininet> h2 wget http://10.0.0.1
--2018-10-29 01:05:13--  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
```

```
mininet> h2 wget http://10.0.0.1
--2018-10-29 01:06:14--  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     33.4KB/s   in 5.2s
```

Experiment 3	Experiment 4
 Packets Seconds 0 20 40 60 80 100 0 10 20 30	 Packets Seconds 0 20 40 60 80 100 0 4 8 12 16
 TCP congestion window (cwnd) timeseries cwnd KB seconds	 TCP congestion window (cwnd) timeseries cwnd KB seconds
 TCP congestion window (cwnd) timeseries cwnd KB seconds	 TCP congestion window (cwnd) timeseries cwnd KB seconds