ACN ASSIGNMENT 1

Priyal Thakkar
Entry No:- 2020BSY7607

Part1:- normal bufferbloat Implementation
Part2:- AQM PIE enabled with delay 20ms
Part3:- AQM PIE Enabled and 10 parallel TCP flow

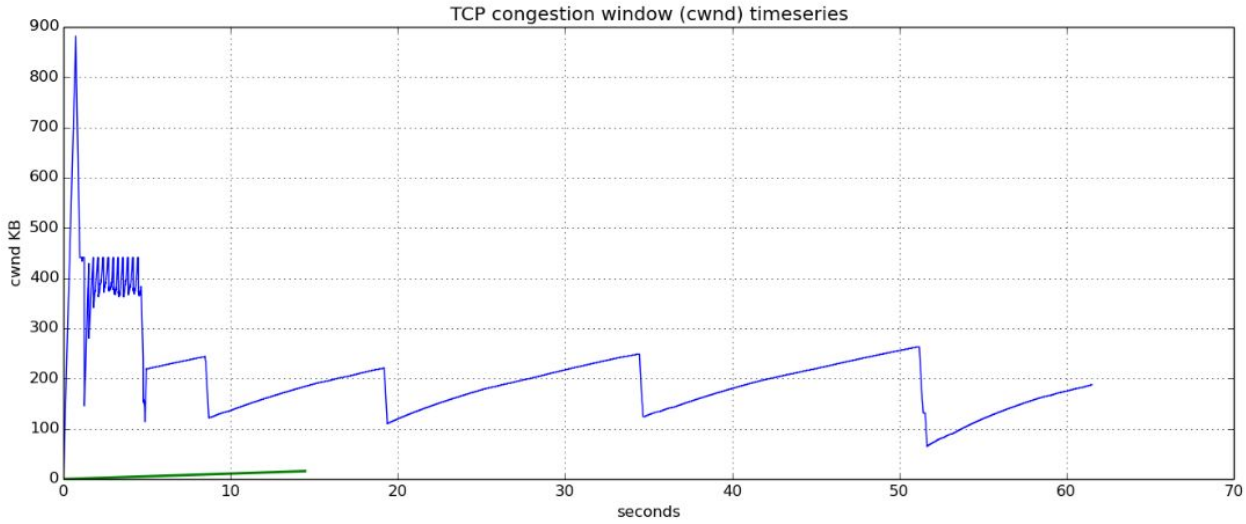
Question 1:-

BUFFERBLOAT IMPLEMENTATION:-

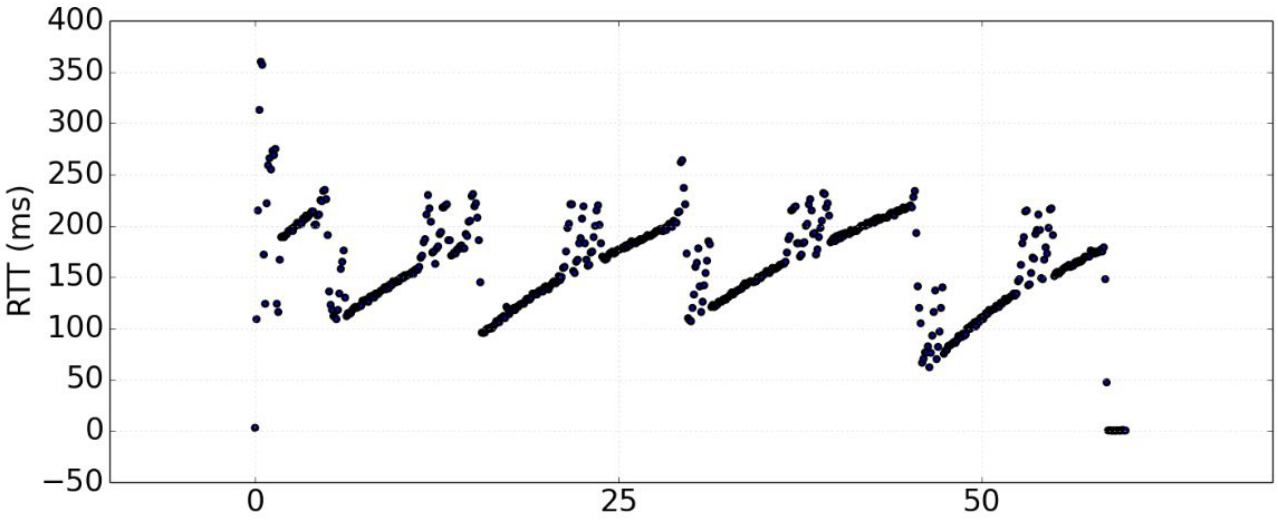
a) *Start a long lived TCP flow using iperf:-*
Command:-
Created server at h2 using command " server = h2.popen("iperf -s -w 16m") "
We have called h1 using h2 using command :- "h1.popen("iperf -c %s -t %s > %s/iperf.out" % (h2.IP(), args.time, args.dir), shell=True)"

b) *Sending ping from h1 to h2 110 time :-*
Command :-
h1.popen("ping -i 0.1 %s > %s/ping.txt"%(h2.IP(), args.dir), shell=True)

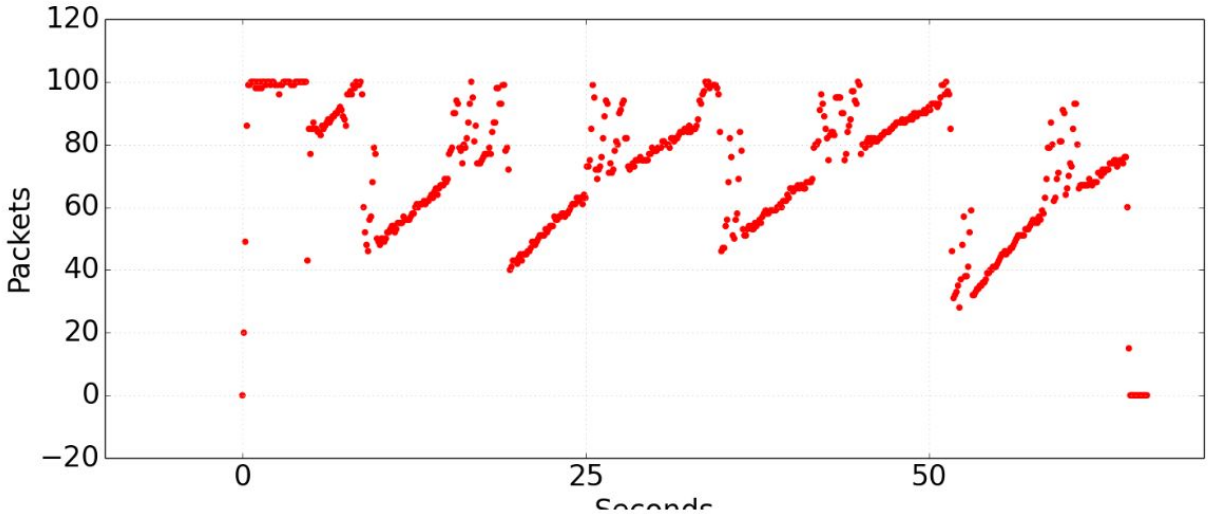
c) *Graph Plotted:-*
   1) *The Long Lived TCP flow cwnd:-*



2) *The RTT reported by ping :-*



3) *Queue Size at bottleneck :-*



(D)

Spawn a webserver on h2. Periodically download the index.html web page using h1 and measure how long it takes to fetch it (on average).

I have started a web server using python/webserver.py.

I have added a loop which runs for 3 cycles and downloads the webpage using curl command.

Command to download web page is :-

```
webpage_time = h2.popen('curl -o /dev/null -s -w %%{time_total} %s/http/index.html' %
        h1.IP()).communicate()[0]
```

Then I have stored time in list and calculated average and concatenated in average.txt file along with standard deviation.

Average: 1.399667

Standard Deviation: 1.196478

(E)

Repeat the above experiment and replot all three graphs after enabling the PIE AQM algorithm on the switch interface attached to the bottleneck link. Set the target delay for PIE to 20 ms

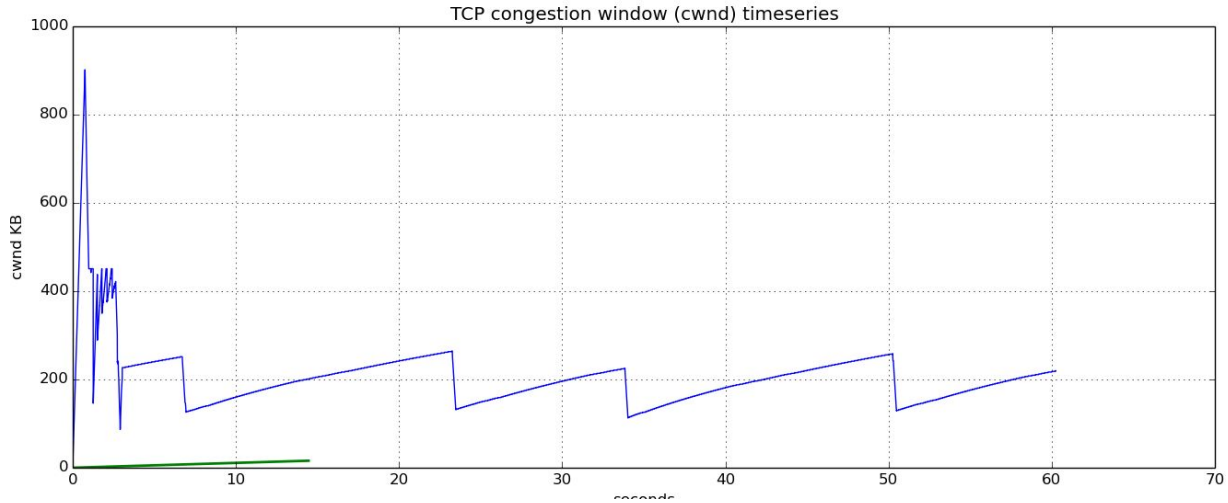
Solution:-

Command used for PIE AQM algorithm :-

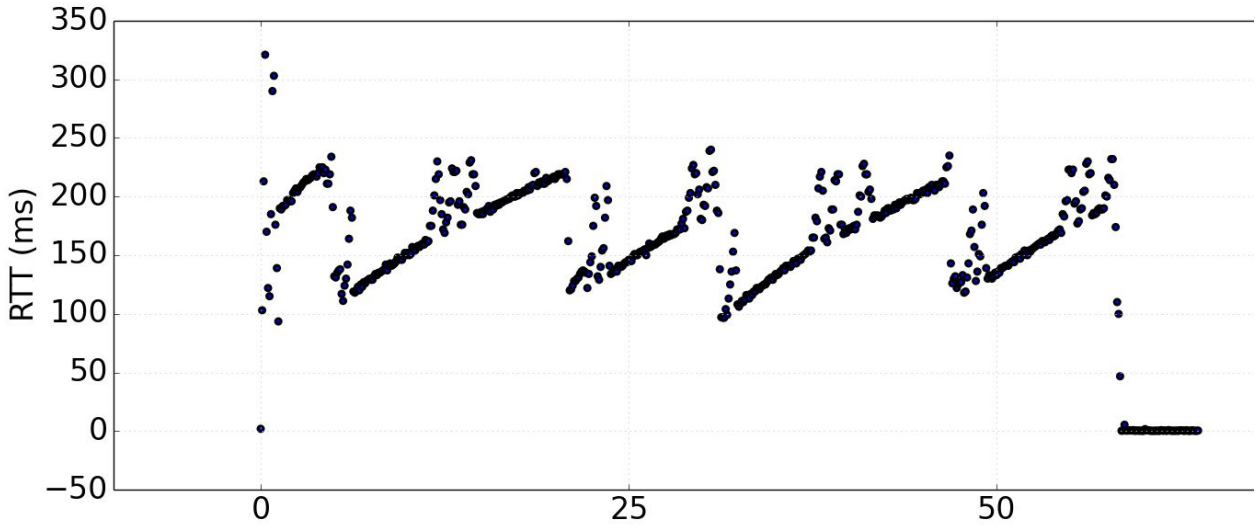
```
switch.cmd('tc qdisc add dev s0-eth2 root pie limit 1000 target 20ms')
print "AQM pie Algorithm added"
switch.cmd('tc -s qdisc show')
```

In this target parameter sets delay to 20m and a pie algorithm is used and the interface that it uses is switched from switch to server.

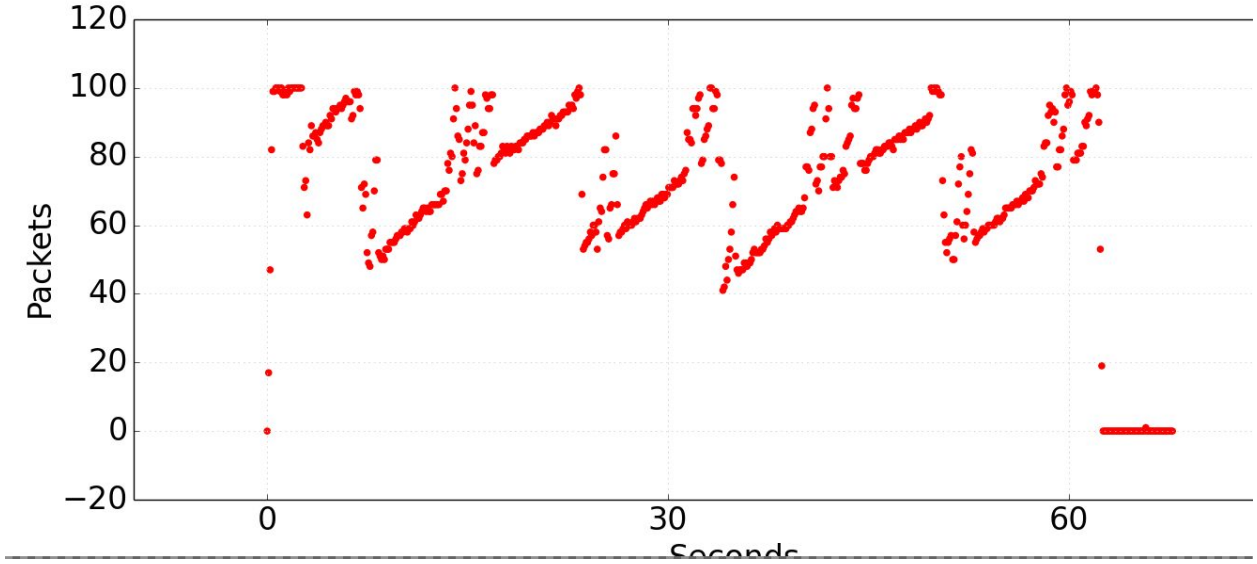
1) *The Long Lived TCP flow cwnd:-*



2) *The RTT reported by ping :-*



3) *Queue Size at bottleneck :-*



Time Required to download web page from server:-
Average: 1.492952
Standard Deviation: 0.956423

As you can see AQM PIE Algorithm drops the packet early, hence , it reduces the size of the queue so we have less queuing delay and more amount of time is required to get a web page from the server.

(F)

Repeat the above experiment with PIE enabled, but this time, start 10 parallel TCP flows with iperf.

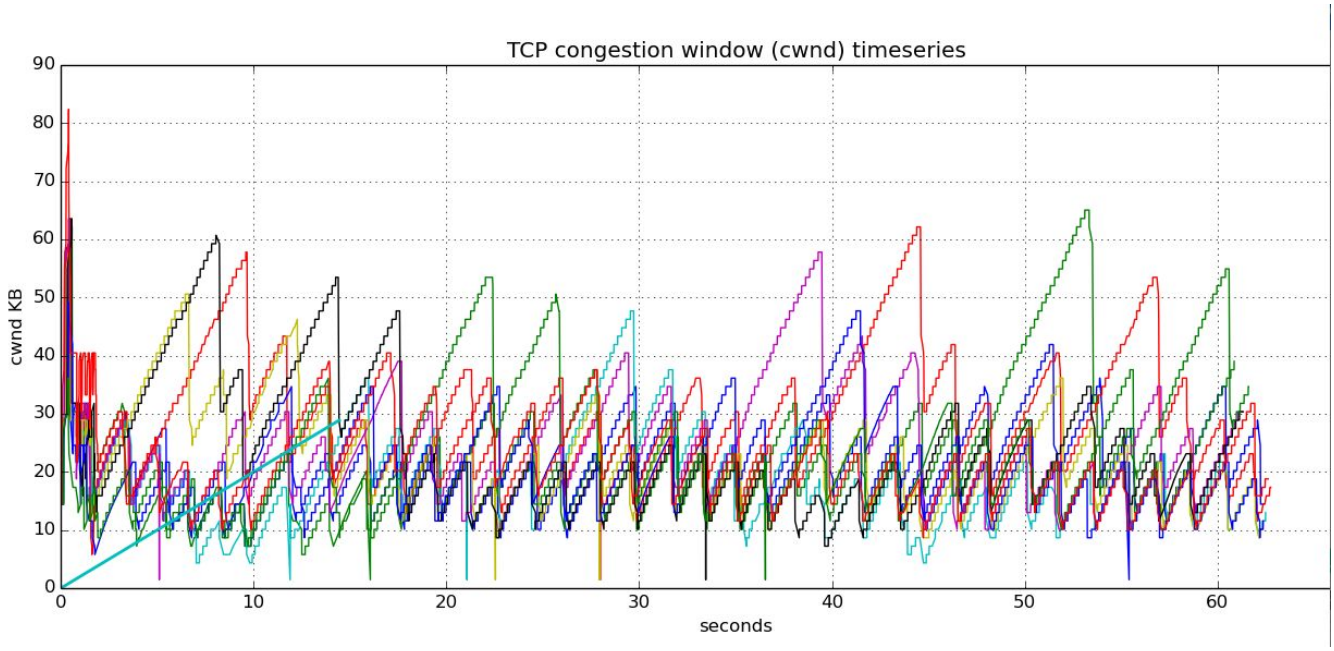
To add 10 TCP parallel Flow we have added -P 10 parameter when host is calling server h2 using iperf command.

Command are as follows:-

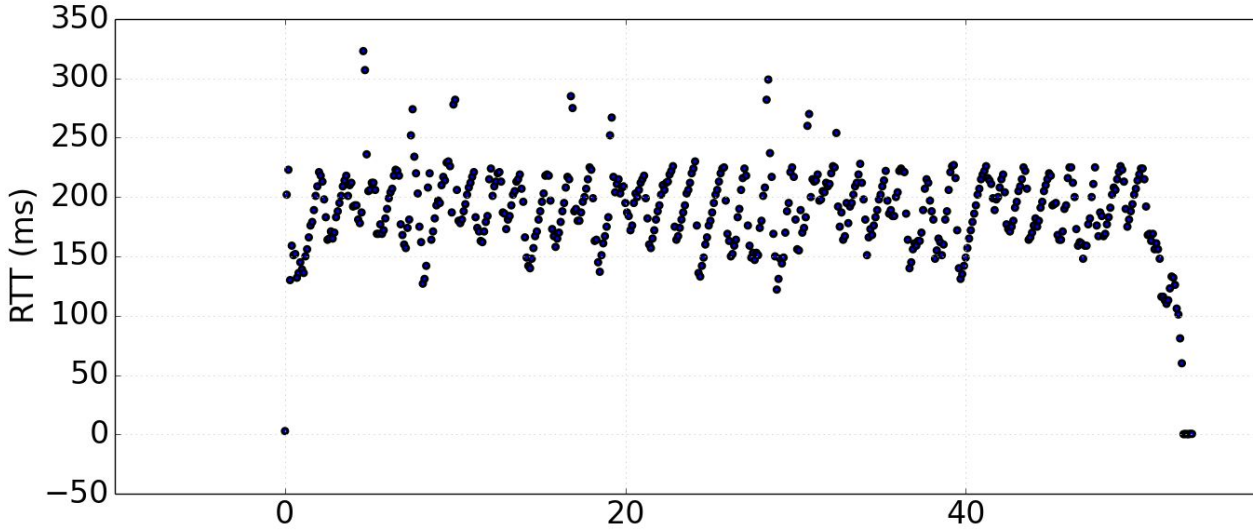
h1.popen("iperf -c %s -t %s -P 10 > %s/iperf.out" % (h2.IP(), args.time, args.dir), shell=True)

You can see different colors in the cwnd graph which states it is used for 10 TCP flows.

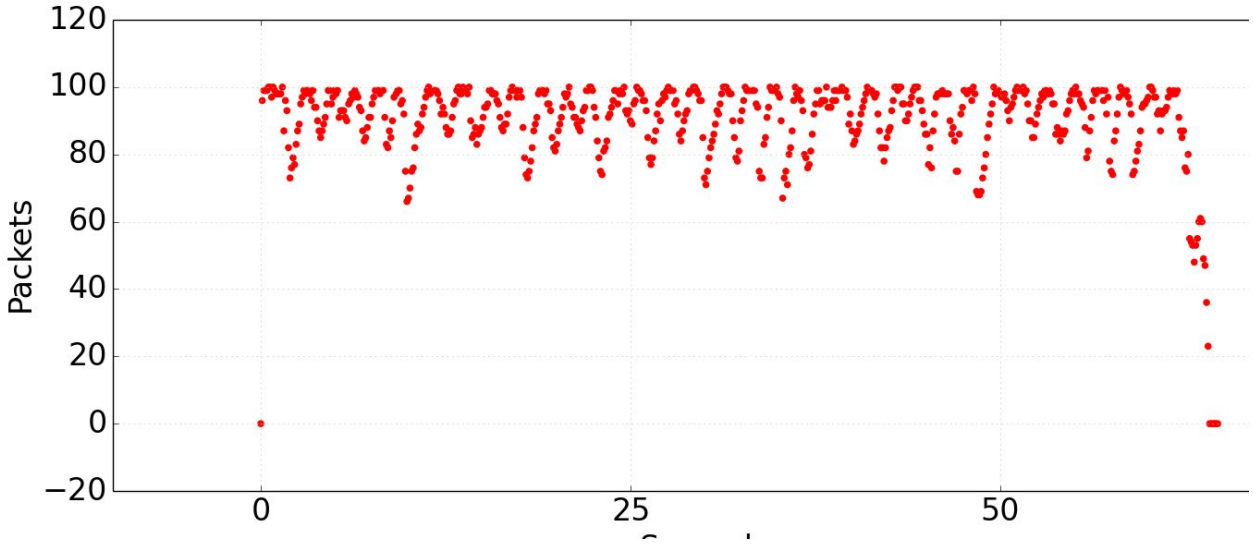
1) *The Long Lived TCP flow cwnd:-*



2) *The RTT reported by ping :-*



3) *Queue Size at bottleneck :-*



Time to Download file from server:-
Average: 2.566133
Standard Deviation: 1.181705

In this graph, we can see that it has more statistical multiplexing performed with 10 parallel TCP links which will help clients to get data faster but due to more packets

being transferred parallely. Hence RTT of each packet increases in comparison to single flow.
Hence, Time required to access a web page from a server alo increases as RTT increases and this you can see with respect to average time to access web pages.

QUESTION 2:-

Assumption :-
We have to demonstrate wifi link i.e. h1-router-switch, so we have considered a link as wireless node and in attached parameter loss = 1% to link which will act like a wifi link with appropriate delay suggested in question1.

Command to add loss ina link:-

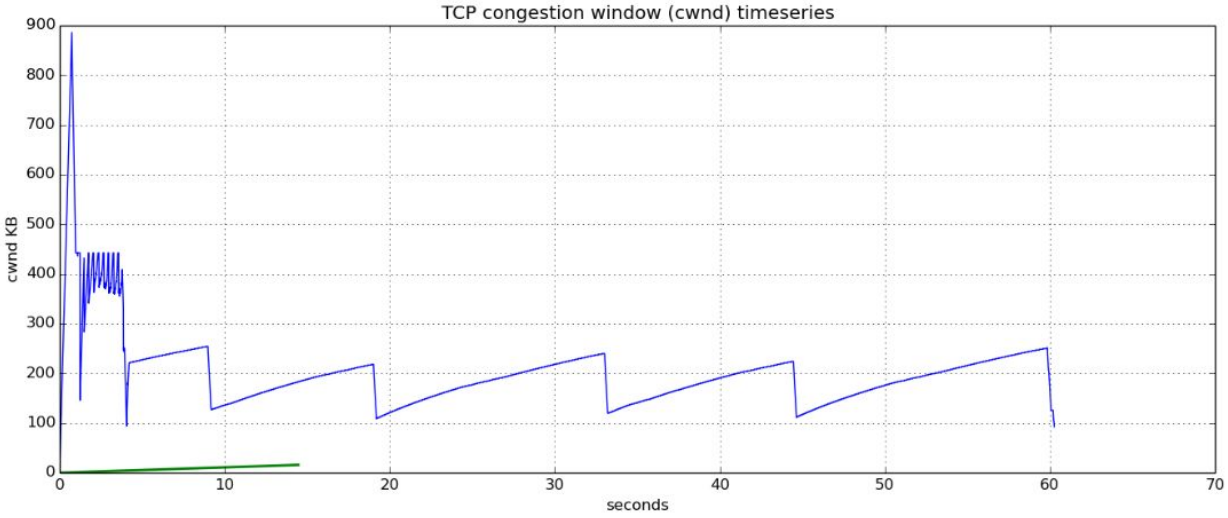
Here i have added loss = 4 i.e. 4 percentage.

```
self.addLink(hosts[i], switch, bw=args.bw_host, delay=args.delay,
        loss = 4,max_queue_size=args.maxq)
```

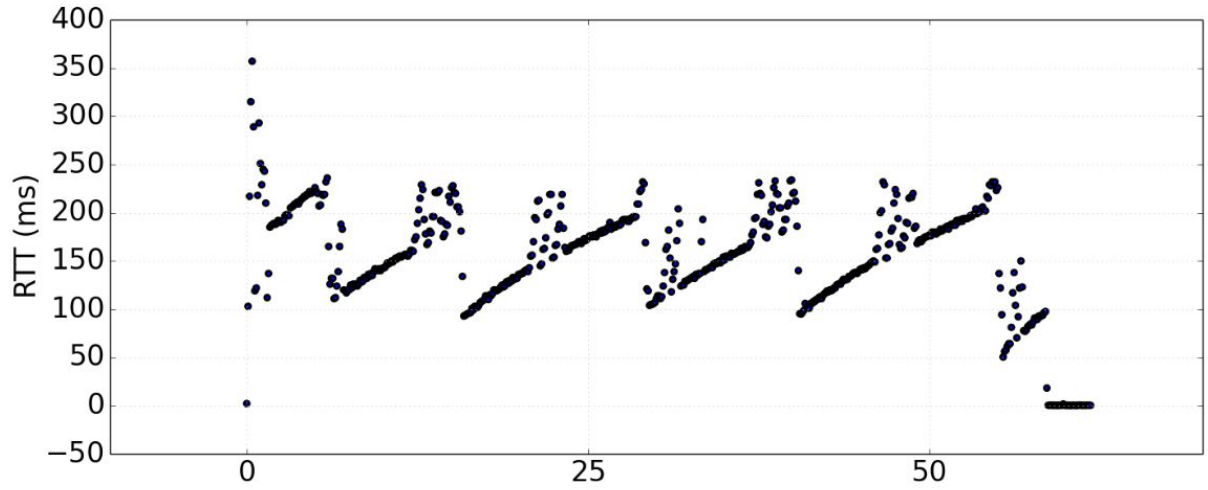
LOSS Value = 0.5

Part1:- normal bufferbloat Implementation

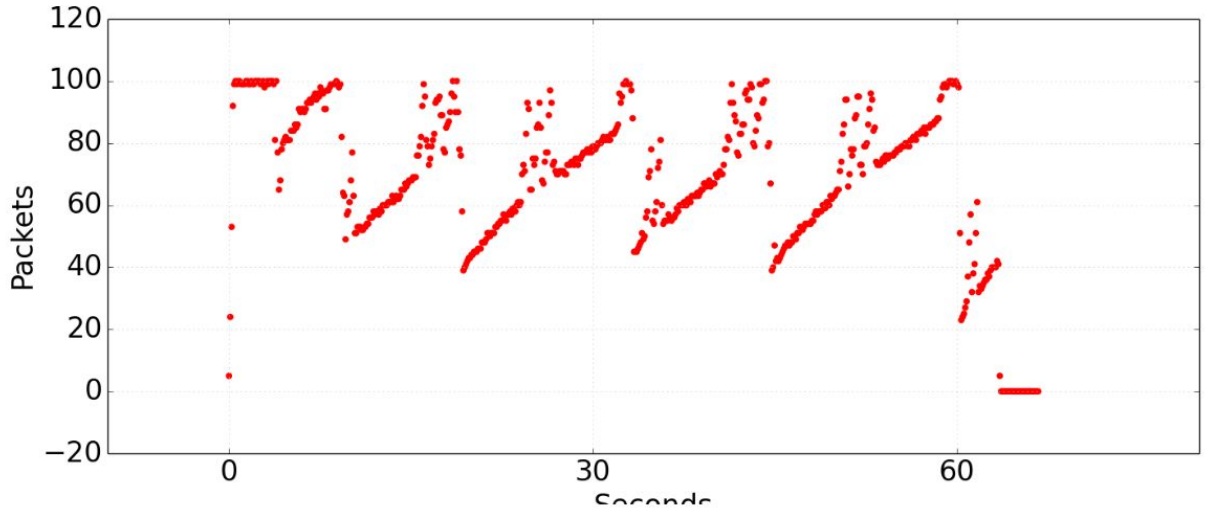
1. *The Long Lived TCP flow cwnd:-*



2. *The RTT reported by ping :-*



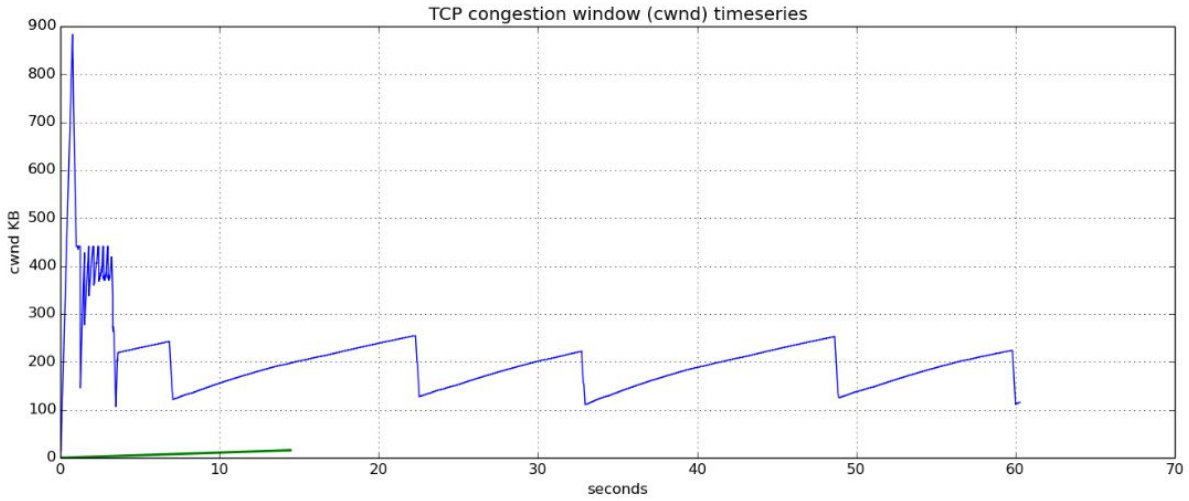
3. *Queue Size at bottleneck :-*



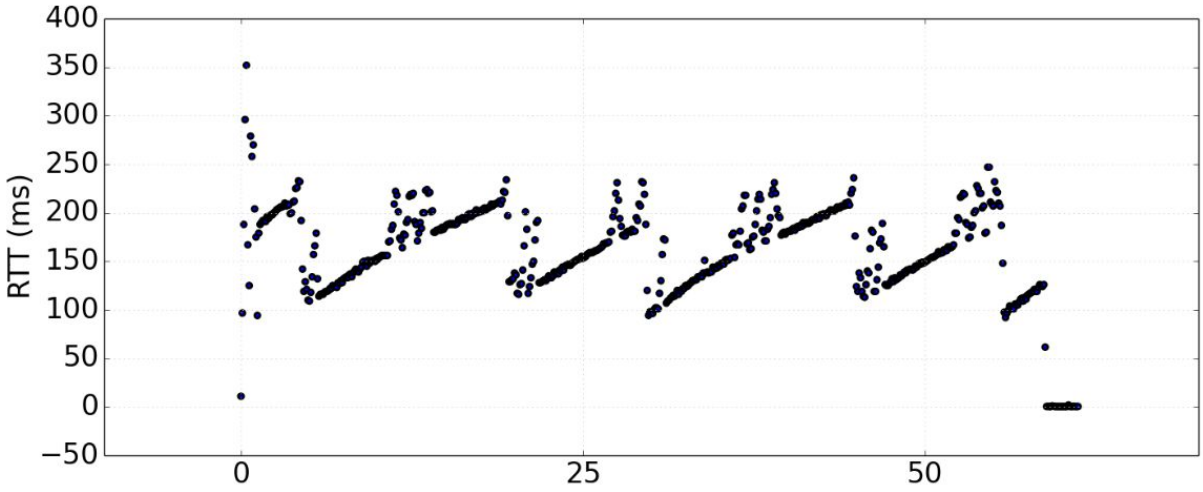
Average: 1.437381
Standard Deviation: 1.302027

Part2:- AQM PIE enabled with delay 20ms

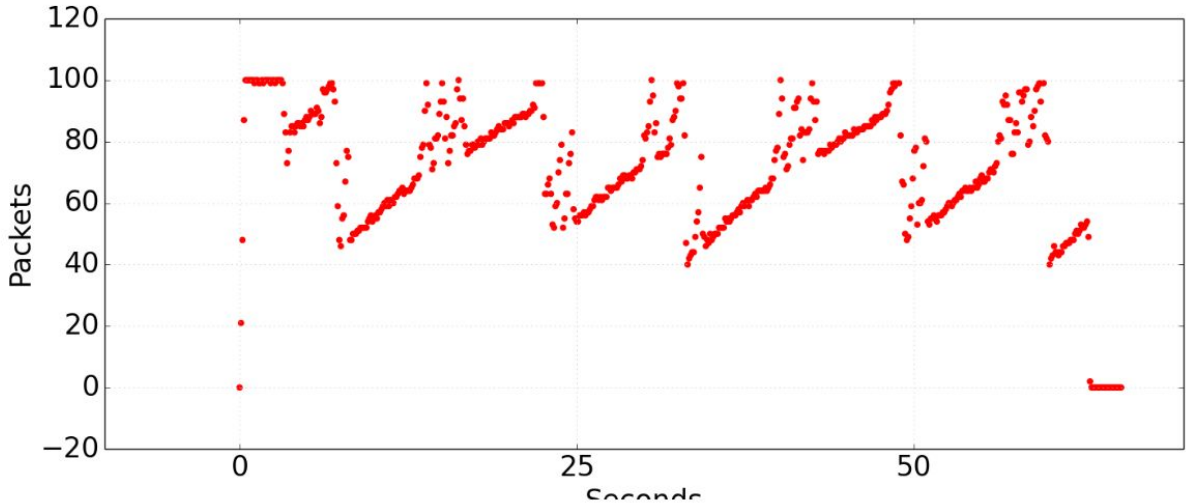
1. *The Long Lived TCP flow cwnd:-*



2. *The RTT reported by ping :-*



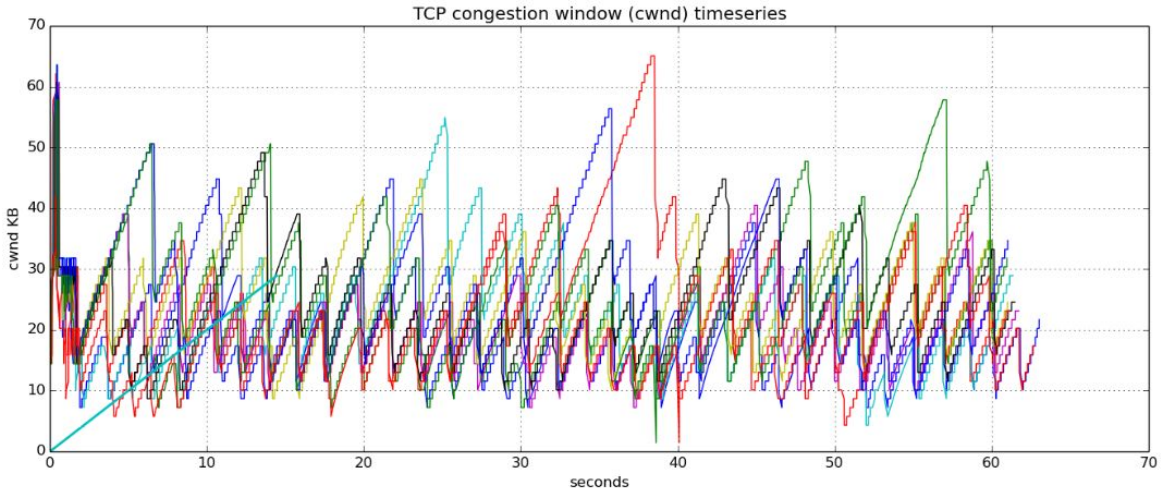
3. *Queue Size at bottleneck :-*



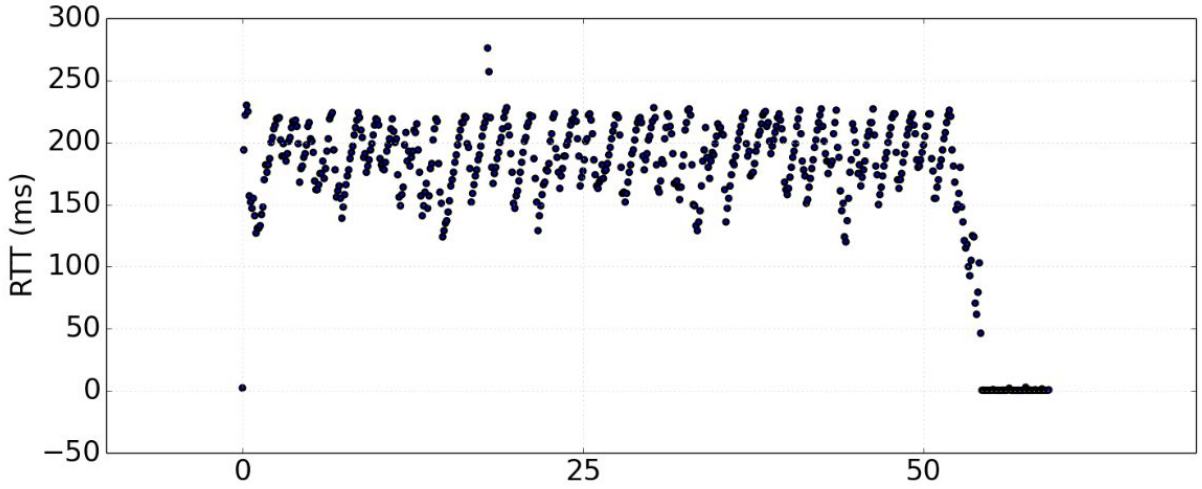
Average: 1.370095
Standard Deviation: 0.878401

Part3:- AQM PIE Enabled and 10 parallel TCP flow

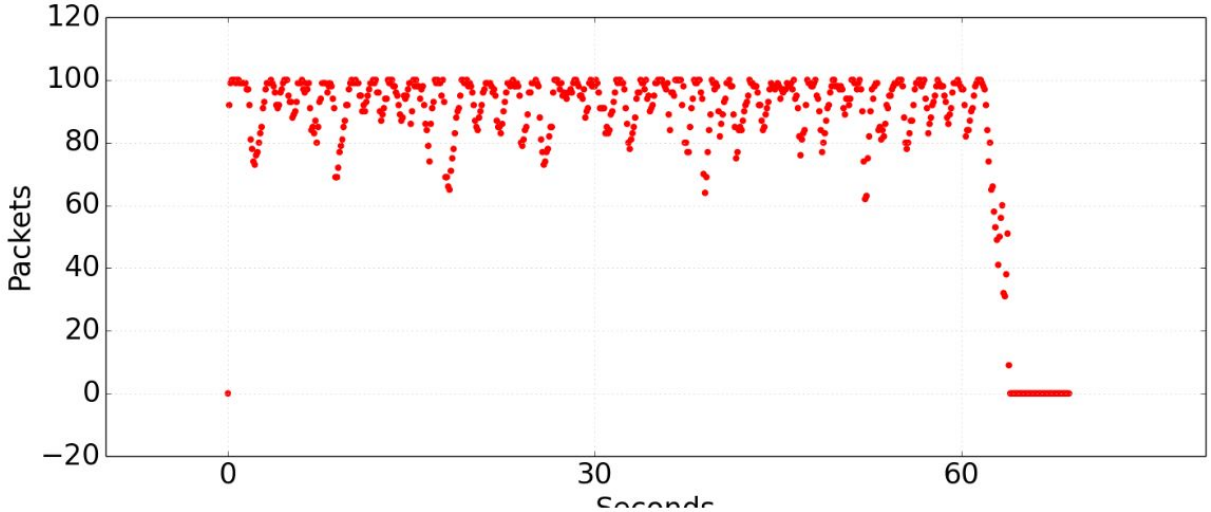
1. *The Long Lived TCP flow cwnd:-*



2. *The RTT reported by ping :-*



3. *Queue Size at bottleneck :-*

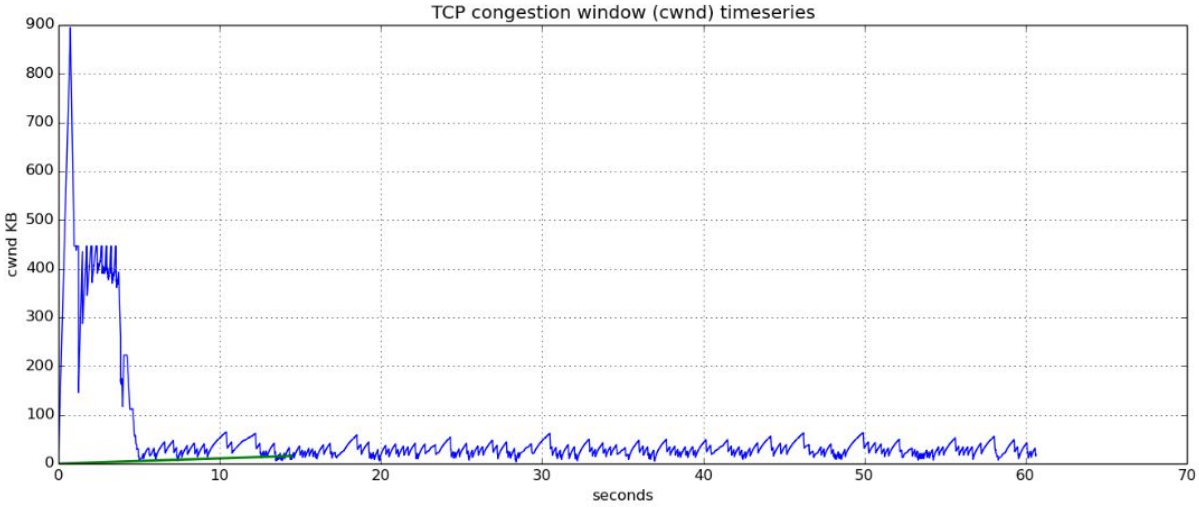


Average: 2.077333
Standard Deviation: 1.078379

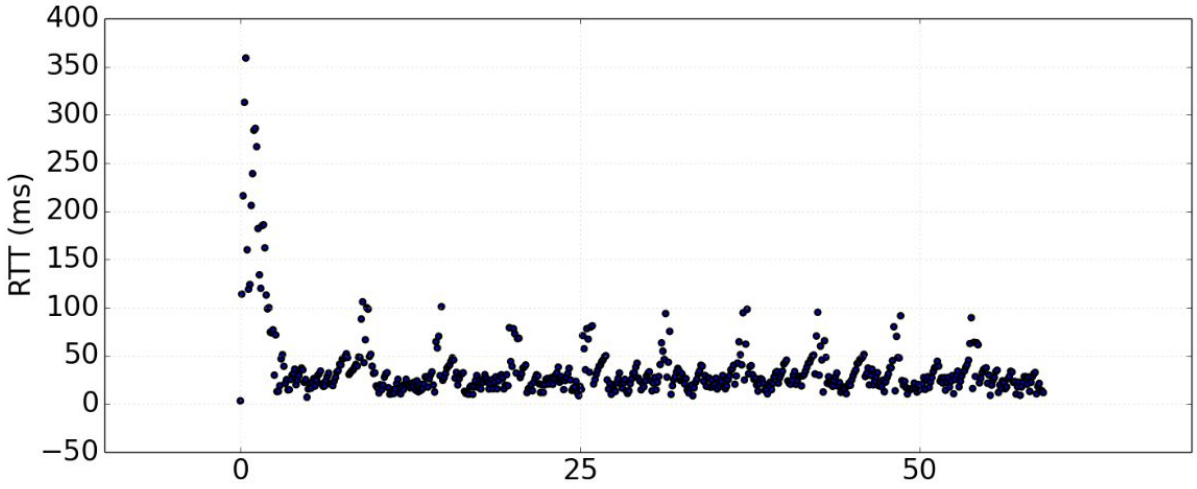
LOSS Value = 1

Part1:- normal bufferbloat Implementation

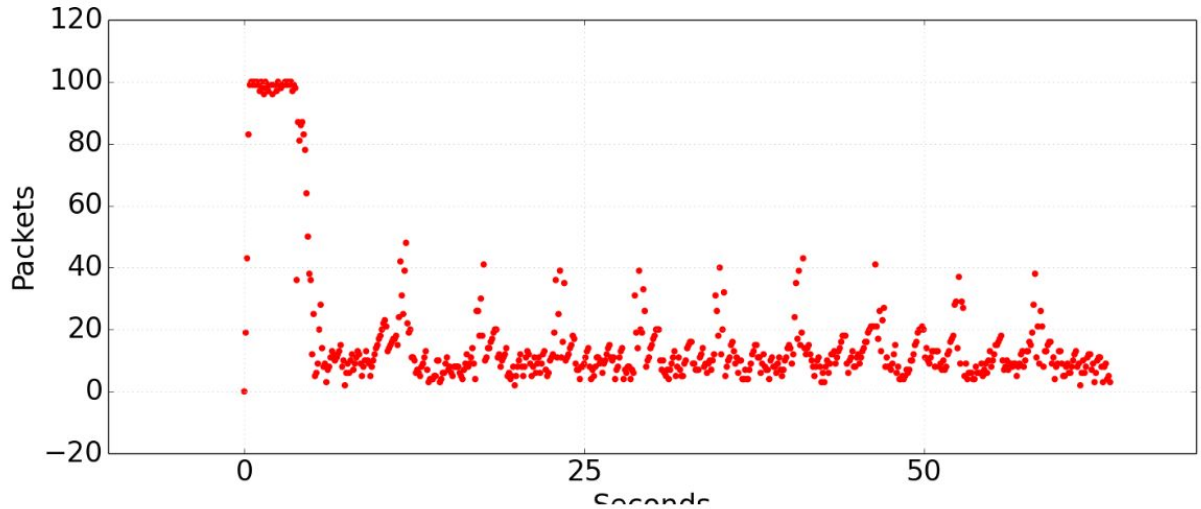
1. *The Long Lived TCP flow cwnd:-*



2. *The RTT reported by ping :-*



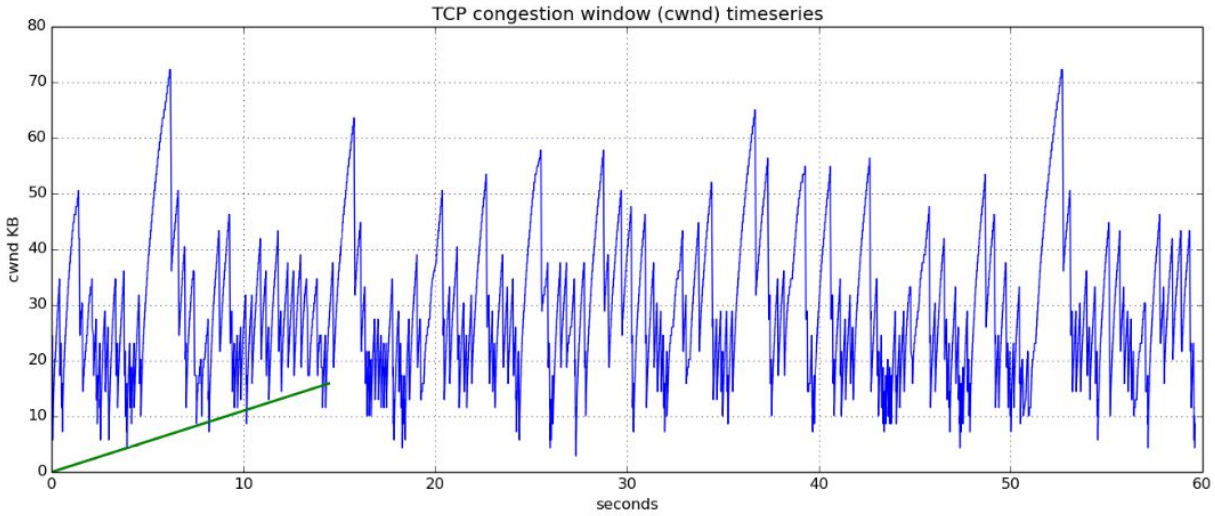
3. *Queue Size at bottleneck :-*



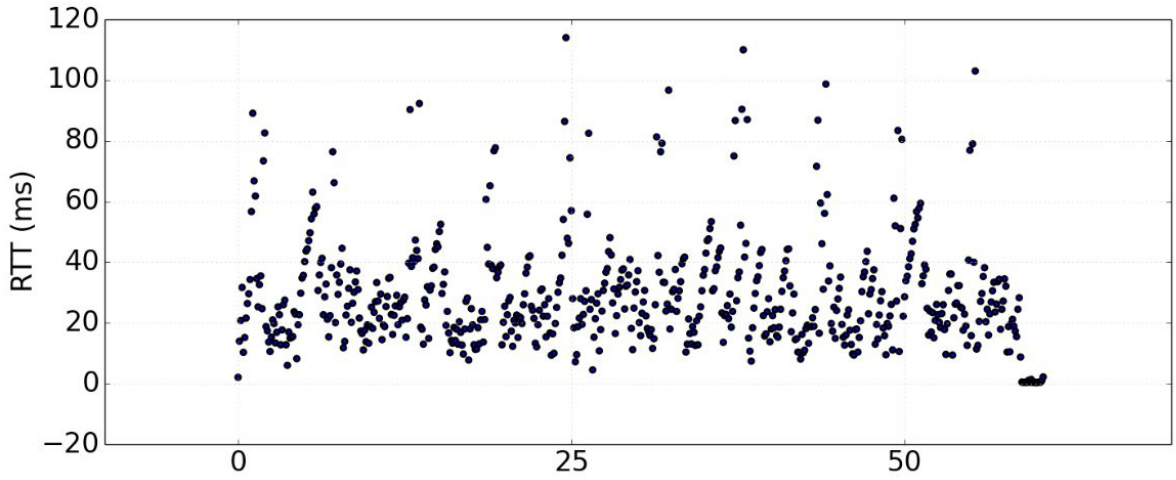
Average: 0.412567
Standard Deviation: 0.660777

Part2:- AQM PIE enabled with delay 20ms

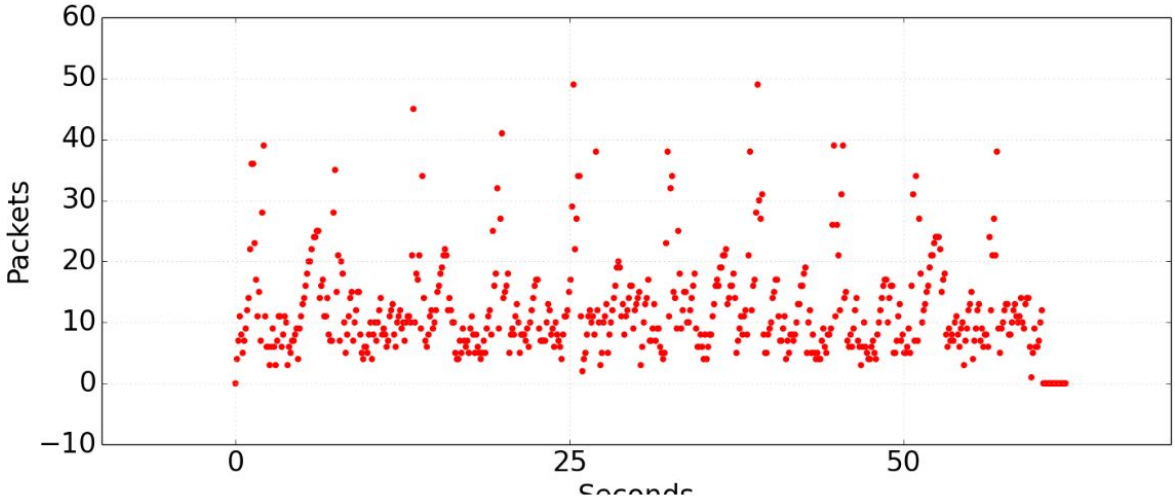
1. *The Long Lived TCP flow cwnd:-*



2. *The RTT reported by ping :-*



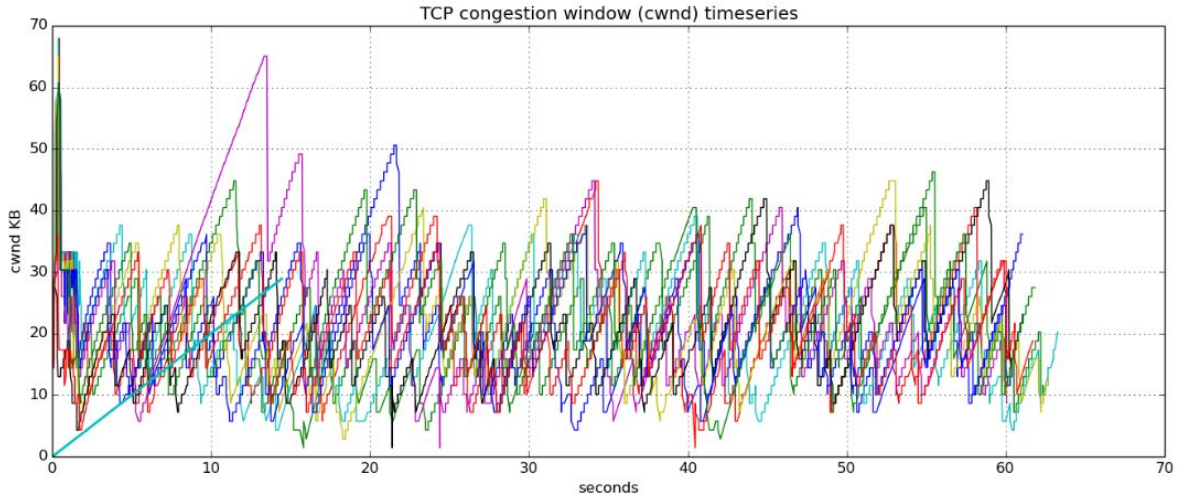
3. *Queue Size at bottleneck :-*



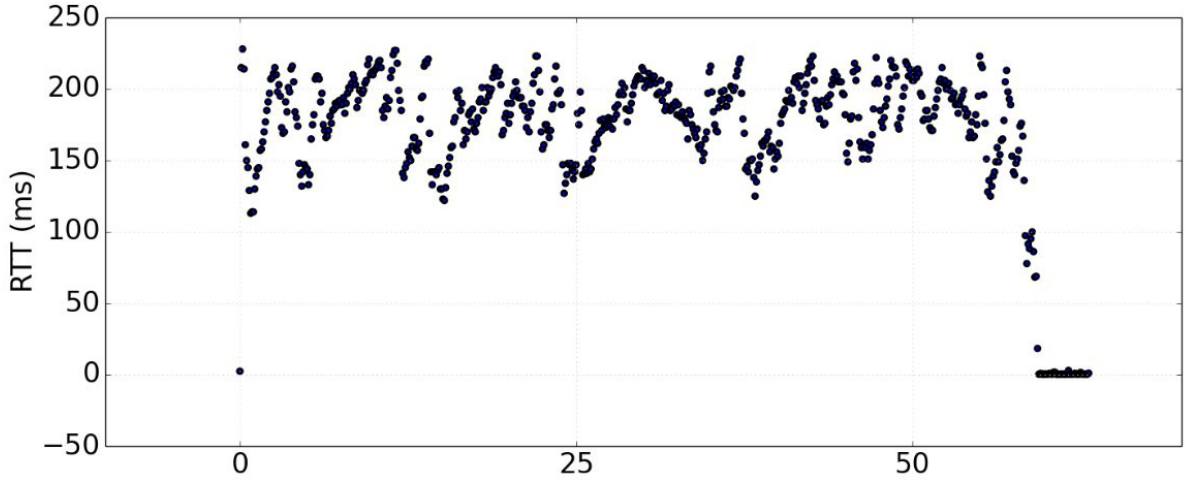
Average: 0.333733
Standard Deviation: 0.194314

Part3:- AQM PIE Enabled and 10 parallel TCP flow

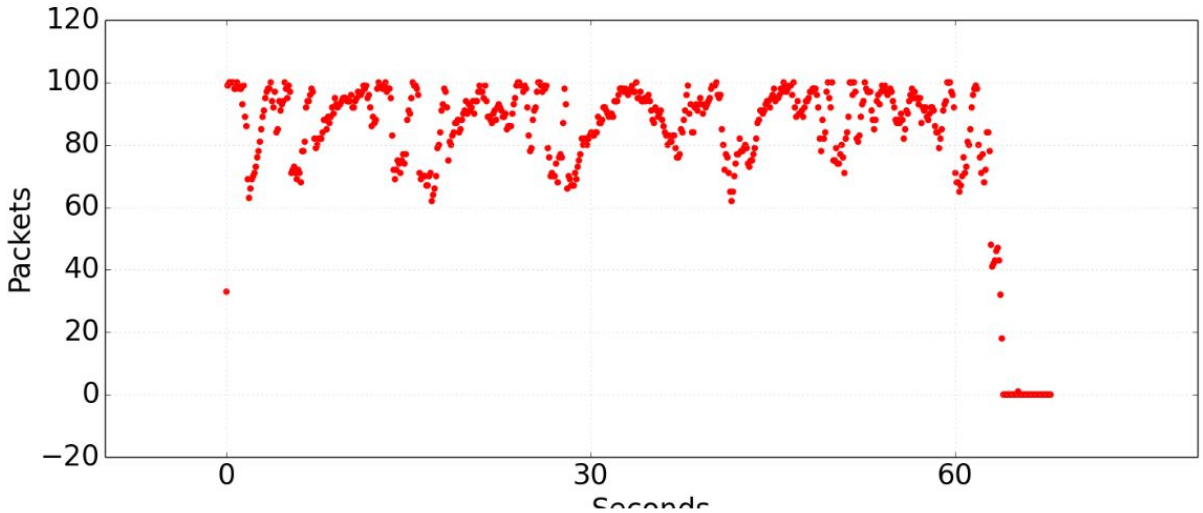
1. *The Long Lived TCP flow cwnd:-*



2. *The RTT reported by ping :-*



3. *Queue Size at bottleneck :-*

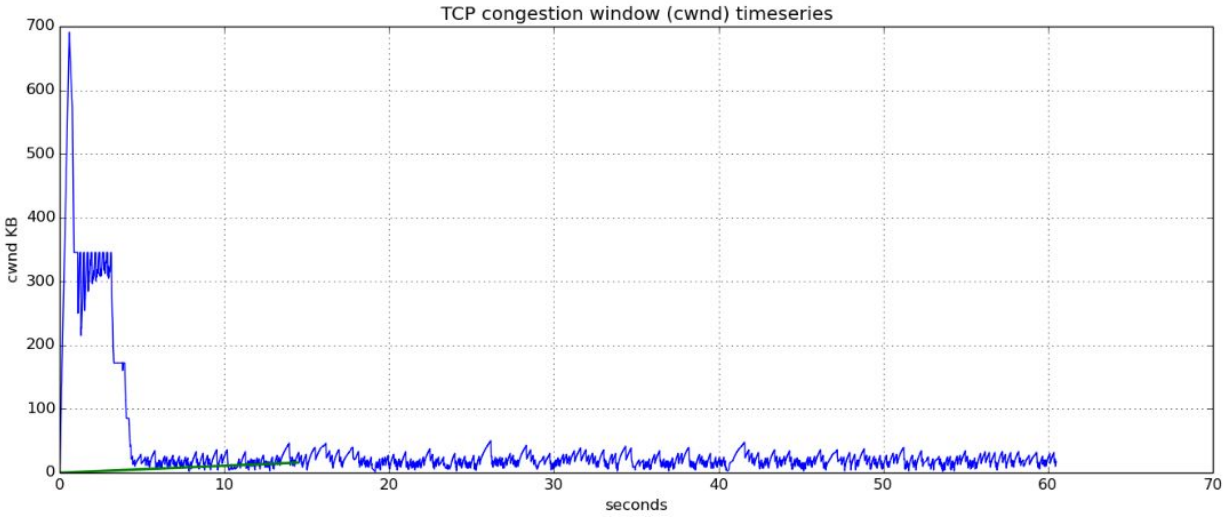


Average: 2.015833
Standard Deviation: 0.731981

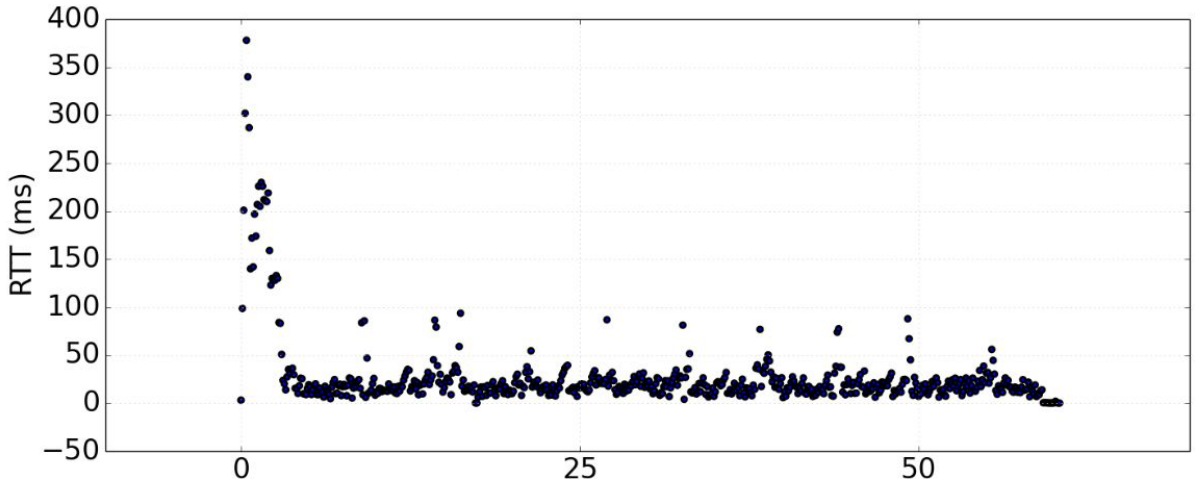
LOSS Value = 2

Part1:- normal bufferbloat Implementation

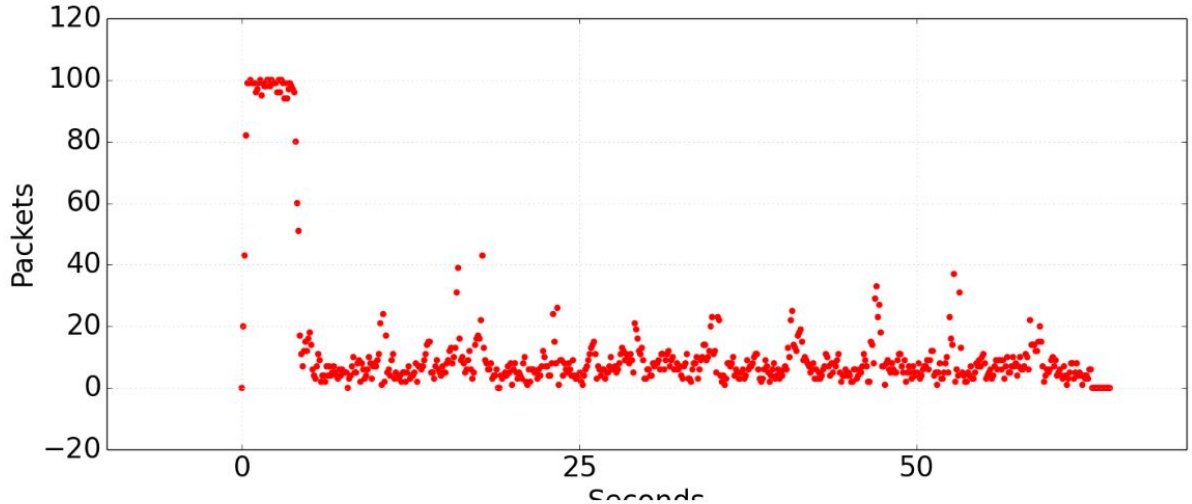
1. *The Long Lived TCP flow cwnd:-*



2. *The RTT reported by ping :-*



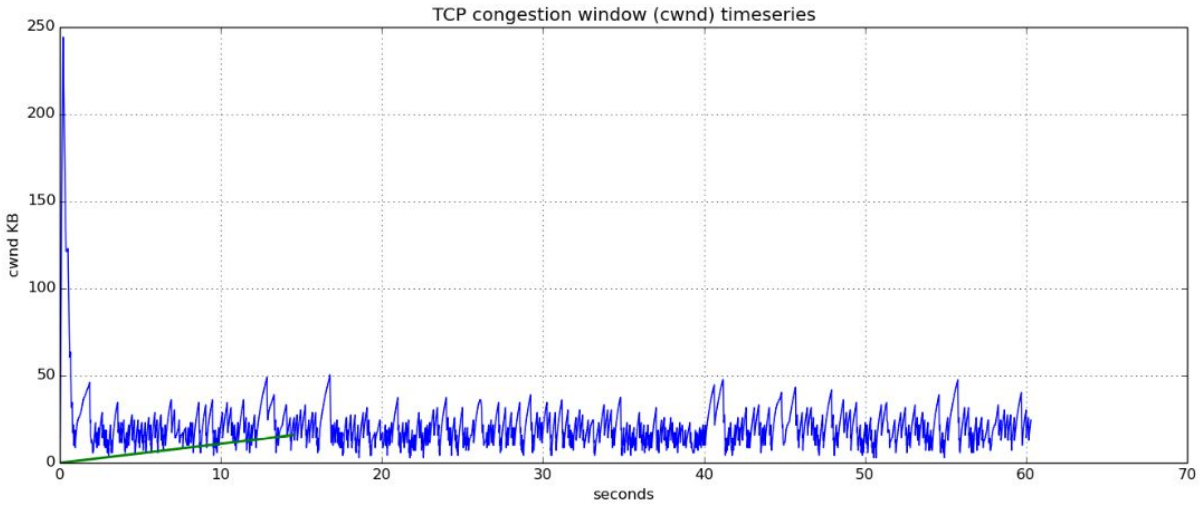
3. *Queue Size at bottleneck :-*



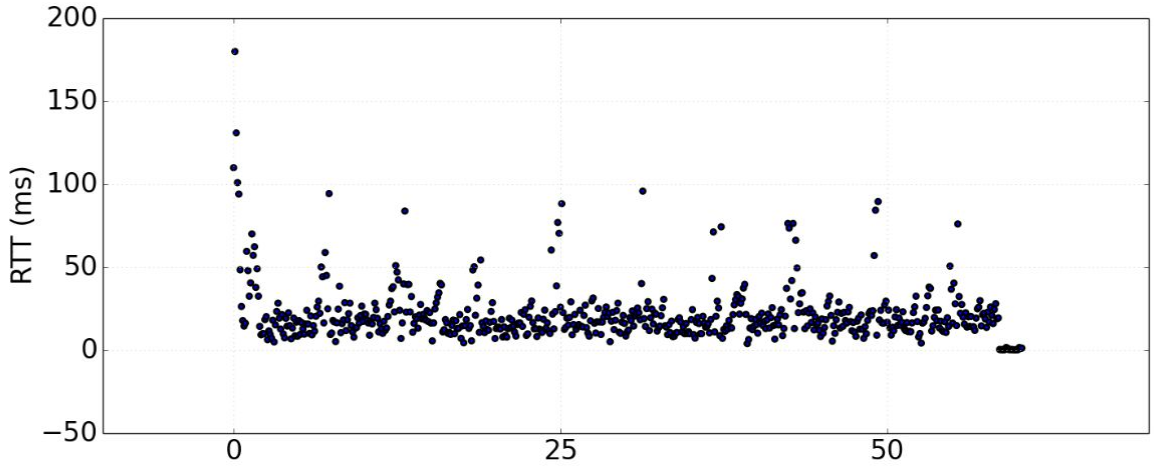
Average: 0.432967
Standard Deviation: 0.618179

Part2:- AQM PIE enabled with delay 20ms

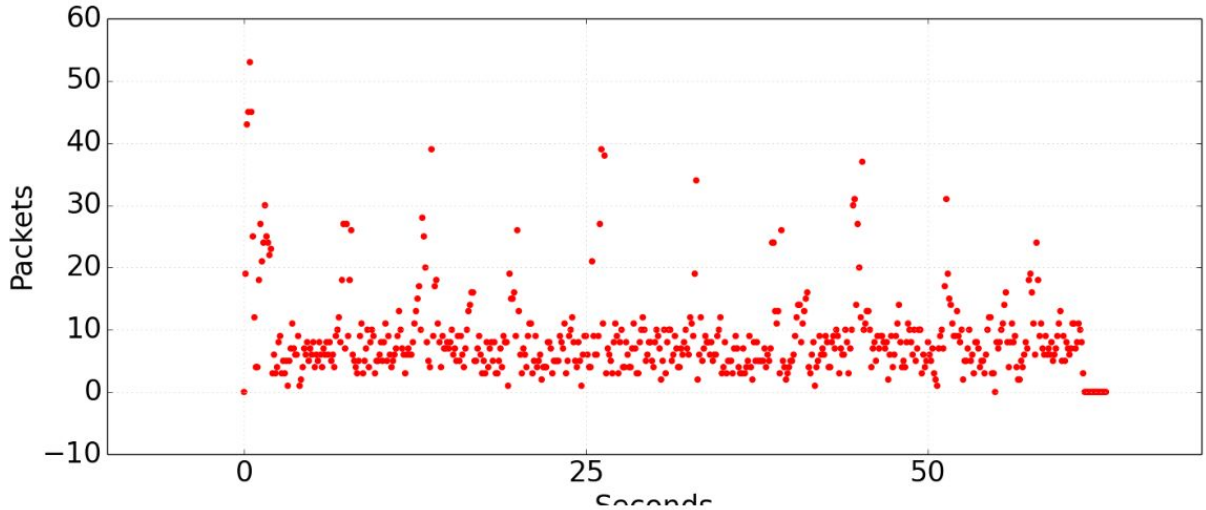
1. *The Long Lived TCP flow cwnd:-*



2. *The RTT reported by ping :-*



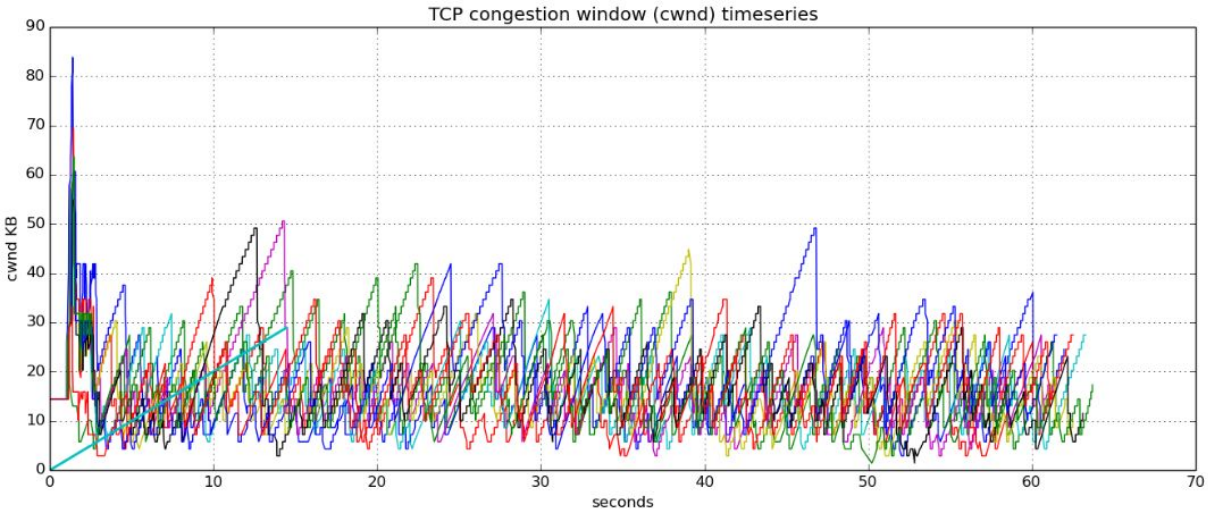
3. *Queue Size at bottleneck :-*



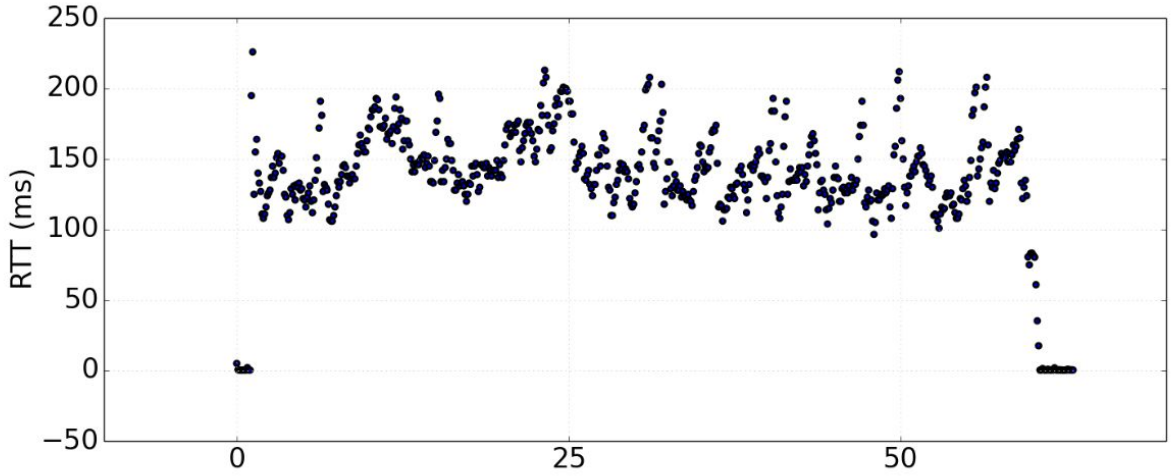
Average: 0.369000
Standard Deviation: 0.258023

Part3:- AQM PIE Enabled and 10 parallel TCP flow

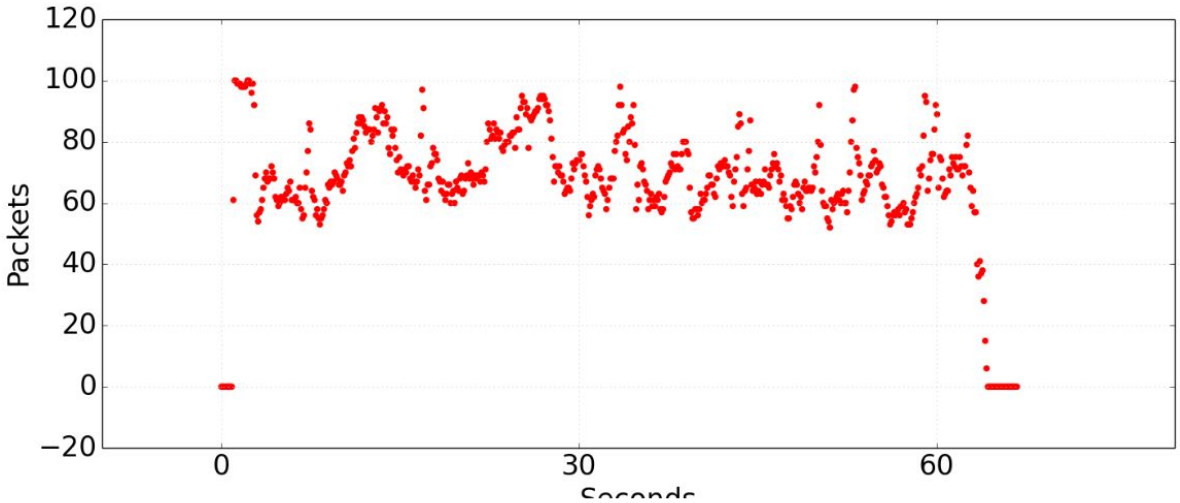
1. *The Long Lived TCP flow cwnd:-*



2. *The RTT reported by ping :-*



3. *Queue Size at bottleneck :-*

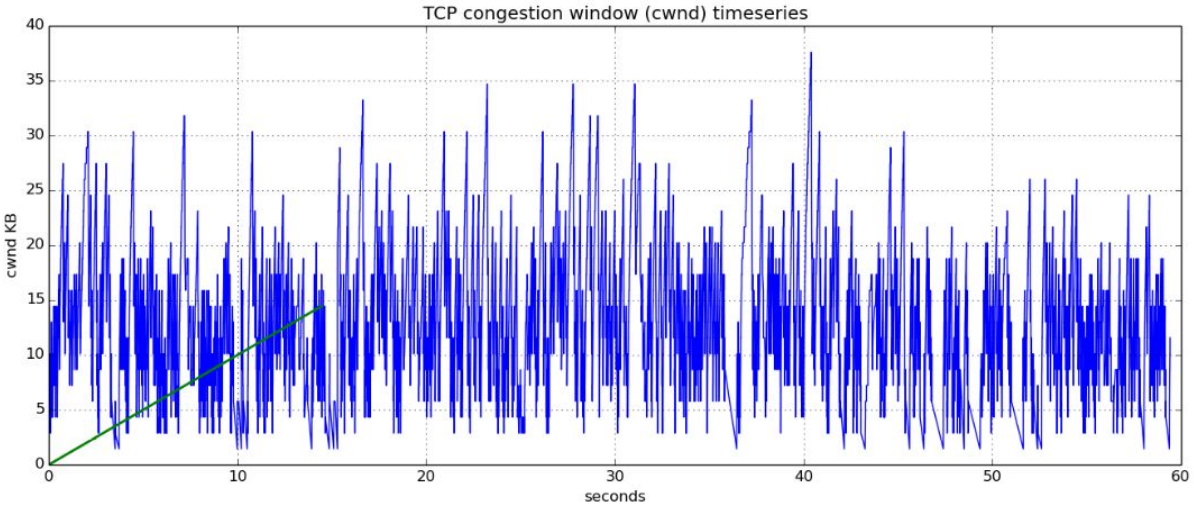


Average: 1.432000
Standard Deviation: 0.722281

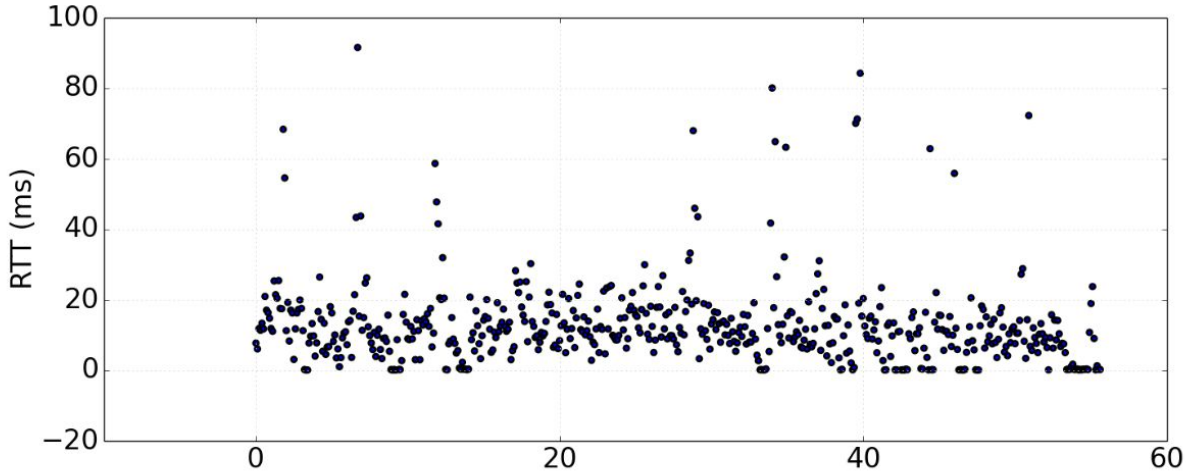
LOSS Value = 4

Part1:- normal bufferbloat Implementation

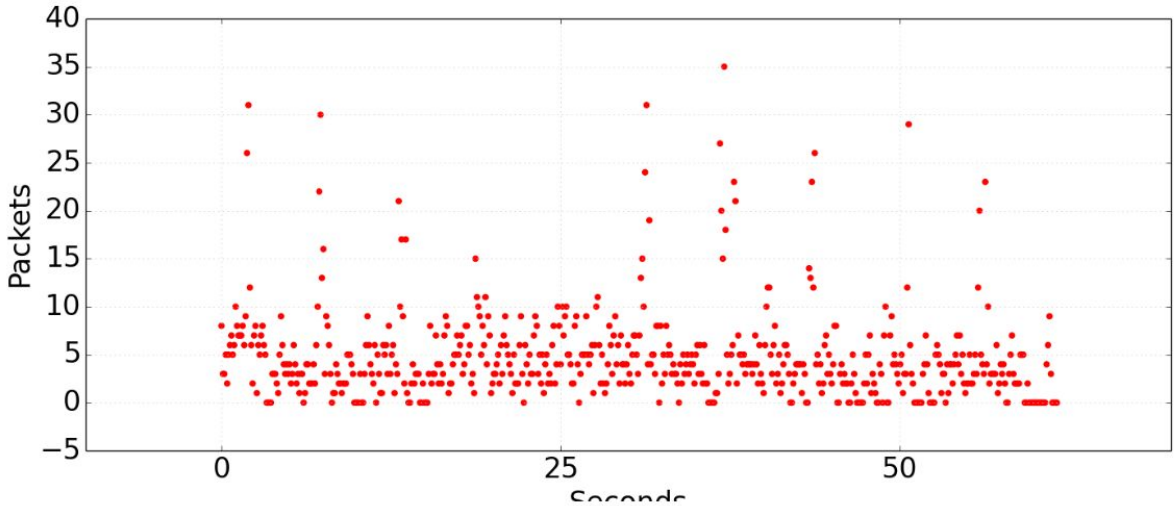
1. *The Long Lived TCP flow cwnd:-*



2. *The RTT reported by ping :-*



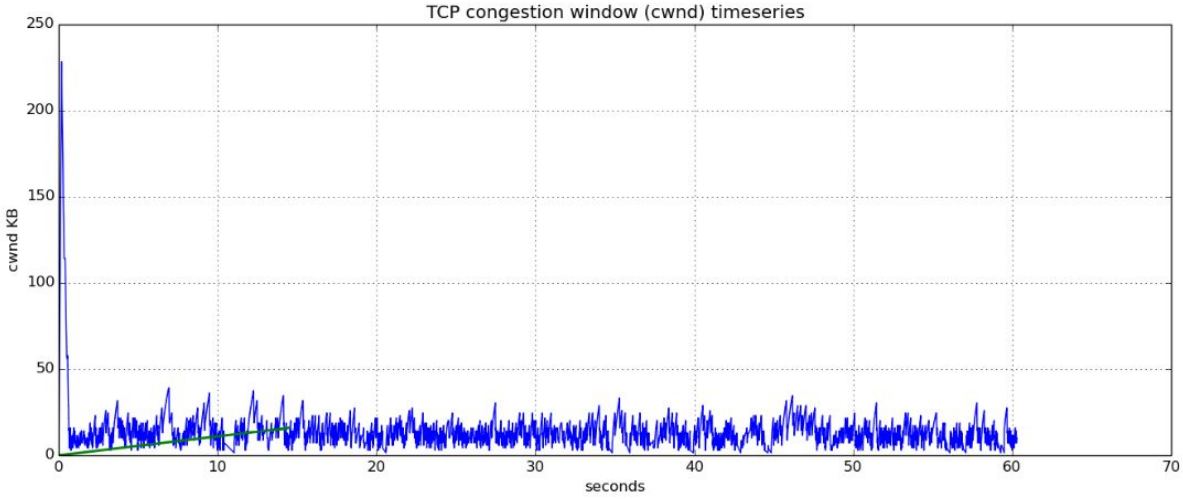
3. *Queue Size at bottleneck :-*



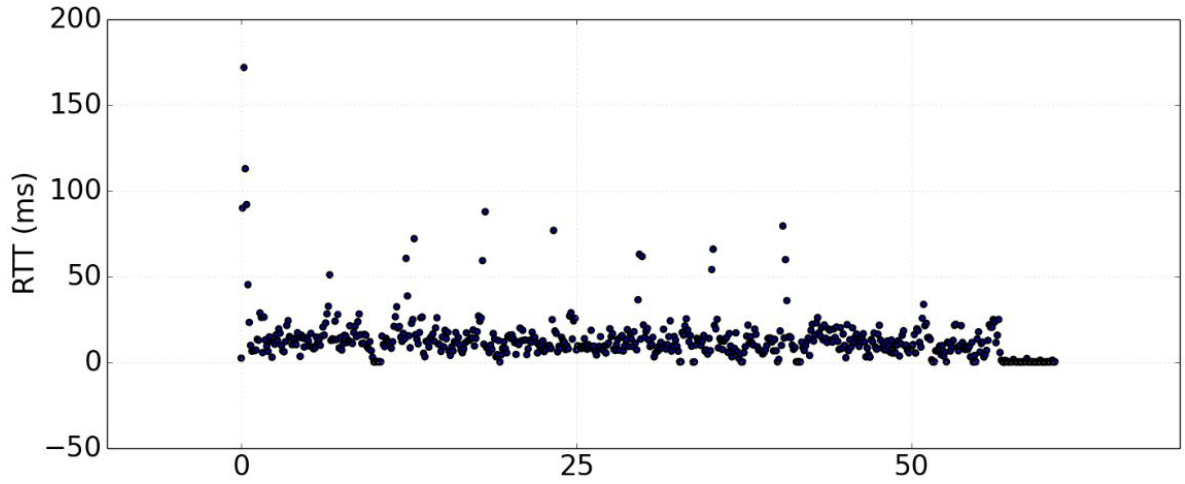
4. *Average: 0.341067*
   Standard Deviation: 0.219077

Part2:- AQM PIE enabled with delay 20ms

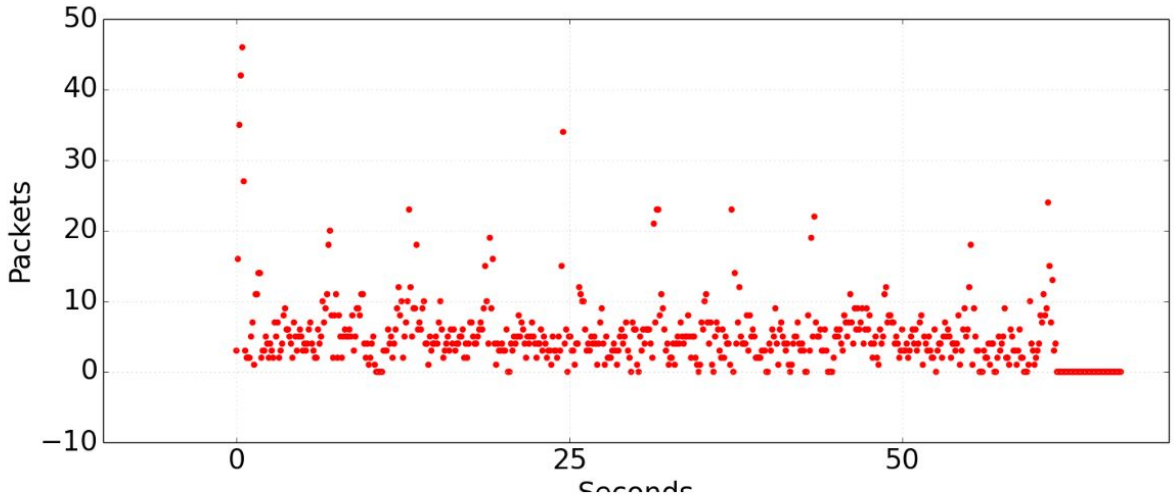
1. *The Long Lived TCP flow cwnd:-*



2. *The RTT reported by ping :-*



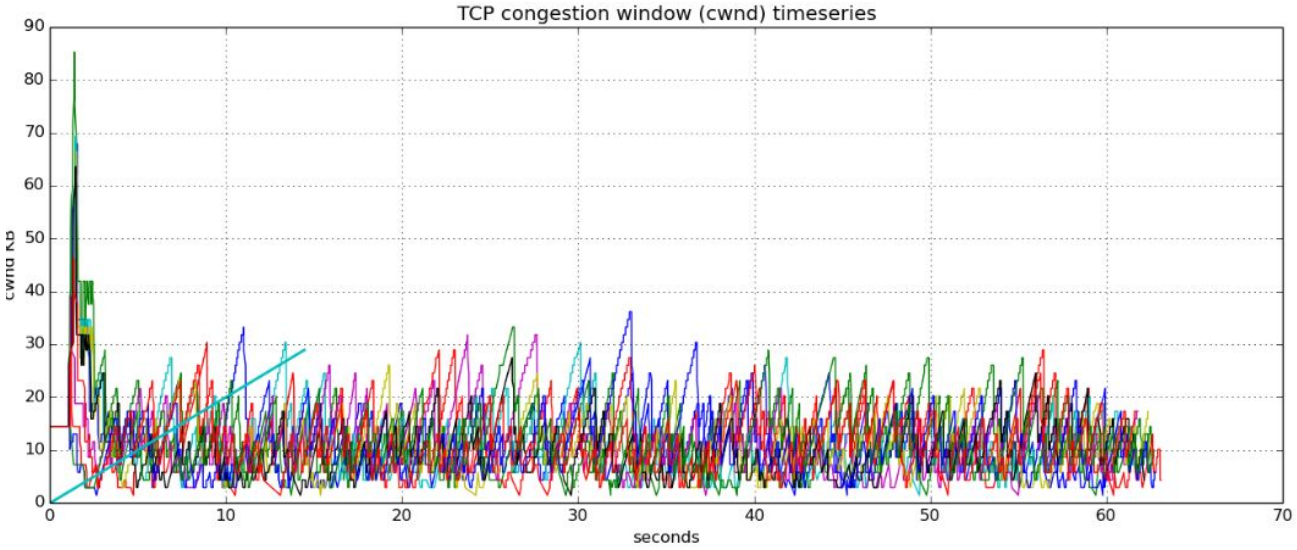
3. *Queue Size at bottleneck :-*



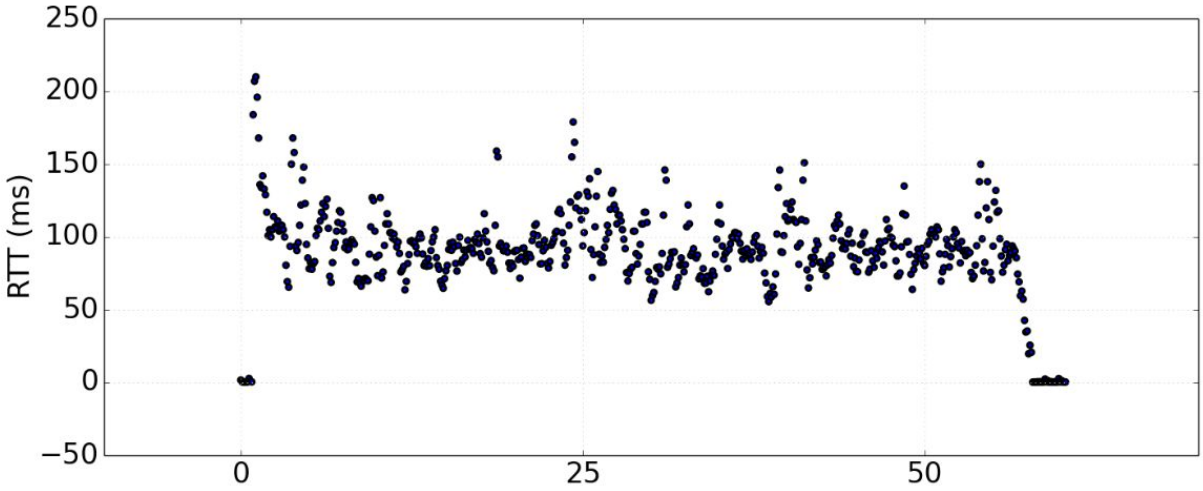
Average: 0.278606
Standard Deviation: 0.181078

Part3:- AQM PIE Enabled and 10 parallel TCP flow

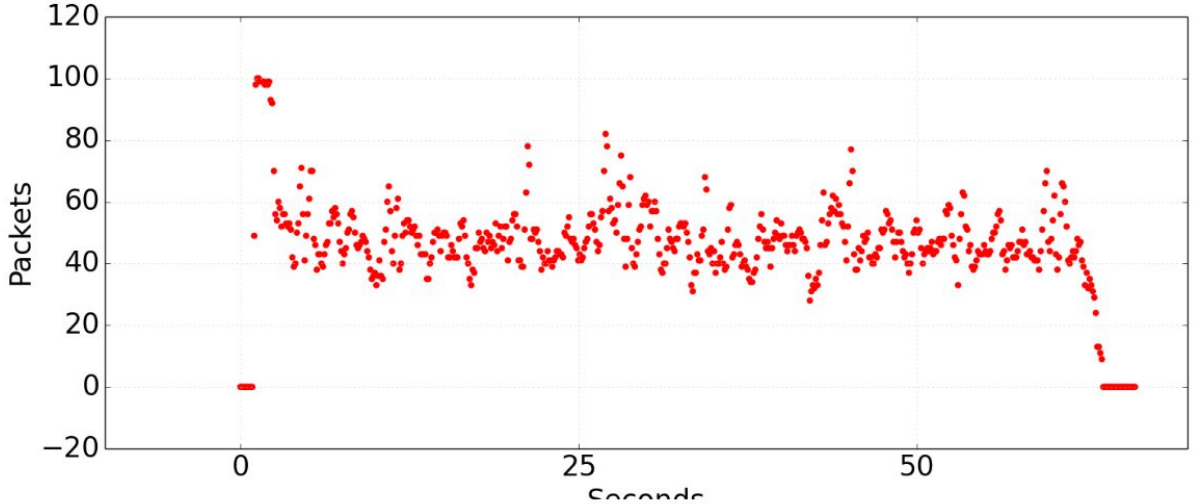
1. *The Long Lived TCP flow cwnd:-*



2. *The RTT reported by ping :-*



3. *Queue Size at bottleneck :-*



Average: 1.013417
Standard Deviation: 0.556601

Observation for Question 2:-

As this question was to understand poor performance due to wireless link:-
We have added loss in our wired link assuming it as a wireless link with noise.
For normal bufferbloat question :-
As noise increases, Cwnd graph is stable for the first few seconds and then it remains below 50KB for all loss percentages . And it is really fluctuating with respect to normal wired links with no loss.

In the RTT graph we can see that as loss increases packets delivery wrt original wired link reduces as there are more number of retransmissions and
In wired link it was around 100 to 150 seconds and after wireless link it is between 0 to 50 and this is due to in wired link, amount of packets are transmitted are more so RTT increases as cwnd size increases and in wireless only 1 packet is transmitted so RTT is less. These observations can be made from the packet queue graph.

In part 2, it follows the same procedure in the CWnd graph as loss increases CWND size reduces gradually.you can see this wrt to graph.
In the queue graph you can see that as loss increases, packet transfer reduces . In loss 0.5 percent it was around 100 packets whereas when you see the graph of 4% loss it is in the range of 0-10 packets.
In the RTT graph we see the same procedure i.e reduction in RTT time as the number of retransmissions increases which you can observe in the cwnd graph.

The part 3 where we have 10 TCP parallel connections you can see that same observation thata CWND size reduces as more number of packets are retransmitted and saame goes with packet and RTT graph.

Question 3:-

Assumption:-

In this part, our assumption is that we have a common server h2 for TCP and UDP(port no:- 5556).
The host h1 asks for packet and video stream from server h2 i.e from different ports of server concurrently.
The video service i.e. UDP is only asked for 20 seconds.
Command used to create UDP server is

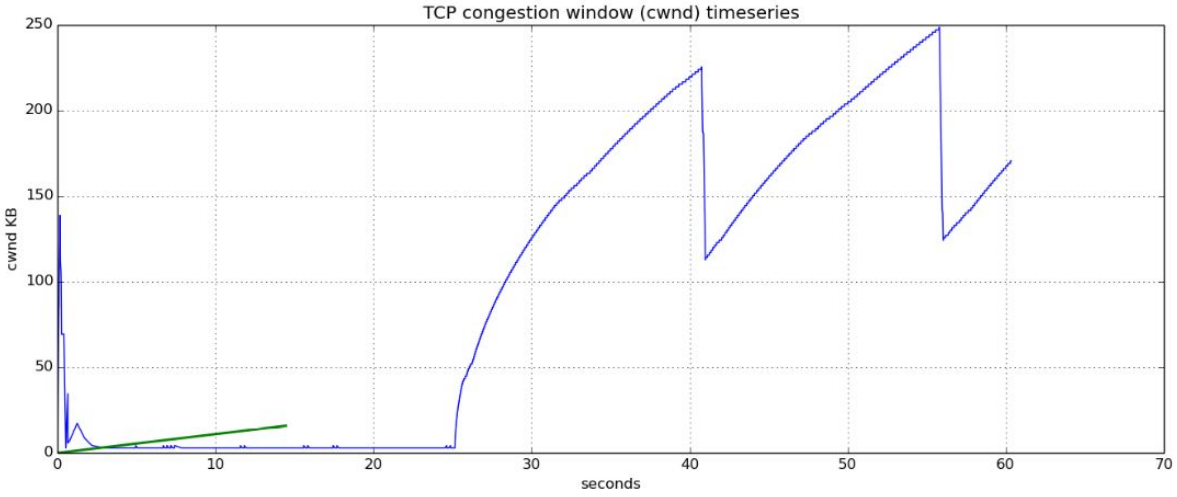
Comparison:-

In the cwnd graph we can see that due to UDP flow , the congestion window size increases at first upto 140 KB but decreases gradually. It remains zero utill 25 seconds because UDP flow is active in that interval. After that interval ,as traffic reduces on client h1 its cwnd size starts gradually increasing as it was in graph1.
In the RTT graph we can see that due to UDP connection, RTT remains constant because UDP flows continuously running with 120 rtt and TCP is only receiving 1 packet per RTT. So all UDp packets are received in that time interval. hence , after that as soon as UDP video stream is stopped. The client h1 regains its time as it was in the original graph. Hence UDP flow effects TCP flow are shown w.r.t. RTT time.
In the packet graph,, we can see that all packets in TCP flow are lost and hence cwnd window remains 11 till UDP flow is active and as you can see only 1 packet is received in that graph.after UDP flow is finishes you can see rise in packet deprived as cwnd size also increases.

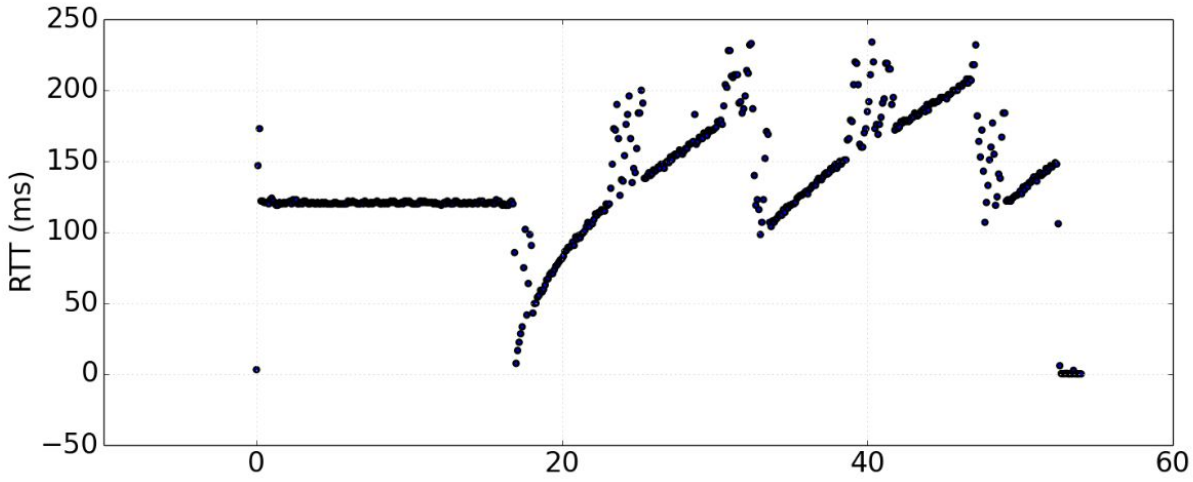
h2.popen("iperf -s -u -p 5566 -i 1 > %s/server.out"%(args.dir),shell =True)

Part1:- normal bufferbloat Implementation

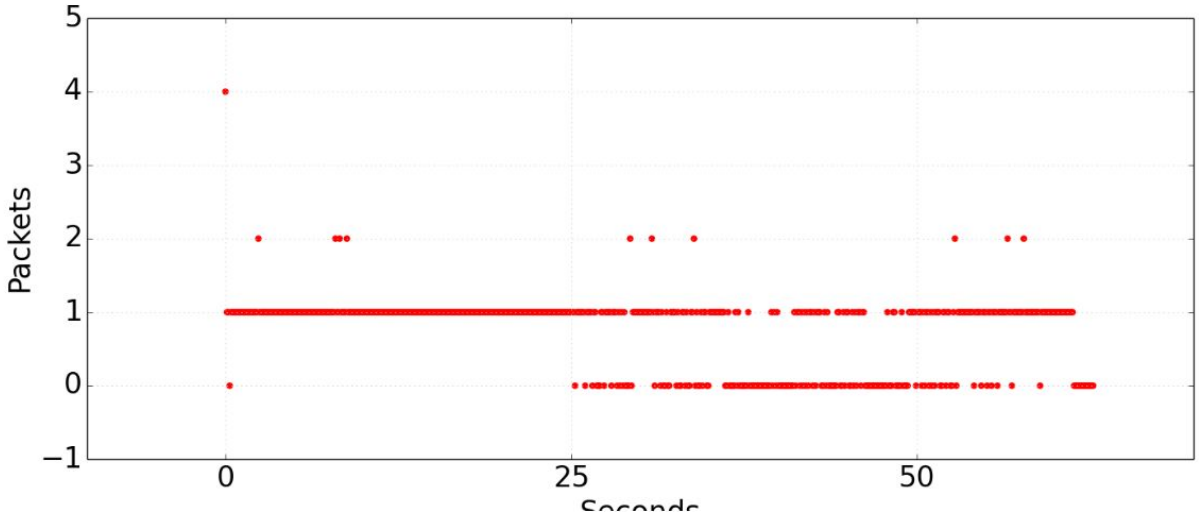
1) The Long Lived TCP flow cwnd:-



2) The RTT reported by ping :-



3) *Queue Size at bottleneck :-*

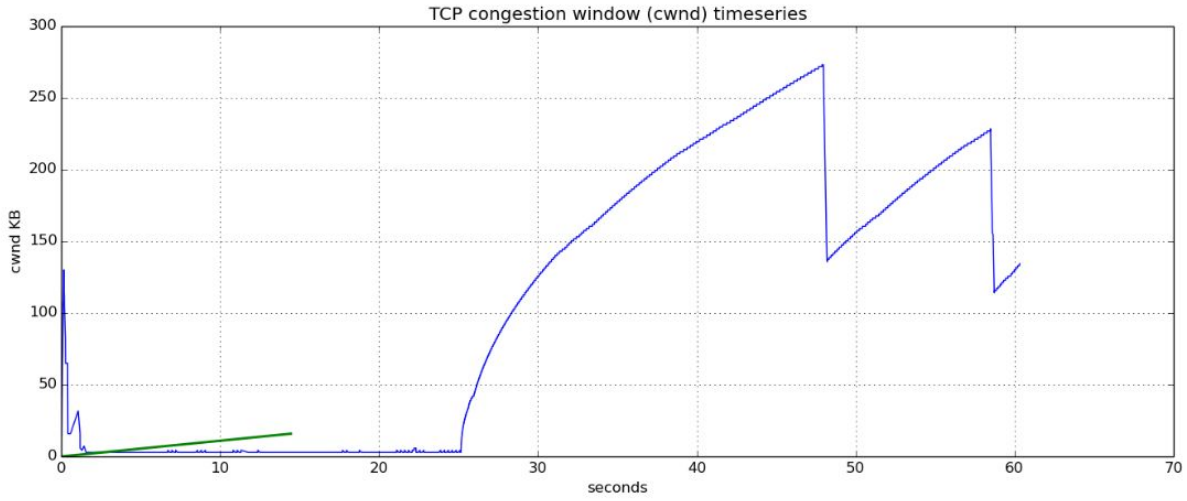


UDP server analysis:-
Jitter = 0.298 ms
Loss = 4.74 %
Latency :- 130ms

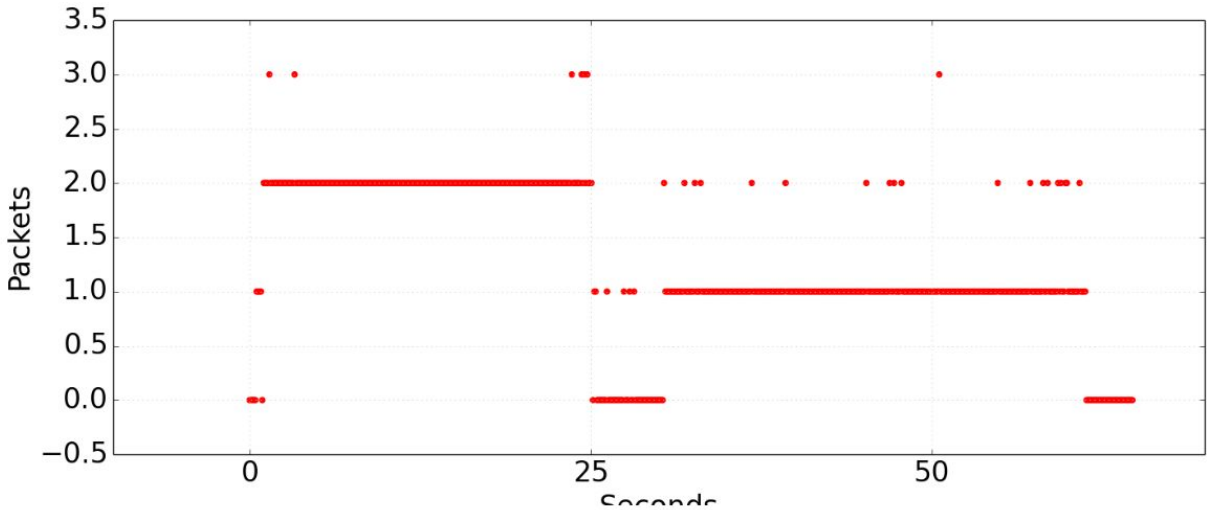
Comparison for Part 2 and part 3:-
In this part 2 and part3 also follows the same structure, until UDP flow is active it affects TCP CWND graph as size remains and as soon as it is finished this graph re-grains its normal shape as it was in question 1.
You can see in this part 2 graph where PIE algorithm is enabled with 20ms and in part where 10 parallel TCP connect is active.
In part 3 you can see that all 1110 parallel connections have cwnd size as 1 and after UDP flow it starts to increase.

Part2:- AQM PIE enabled with delay 20ms

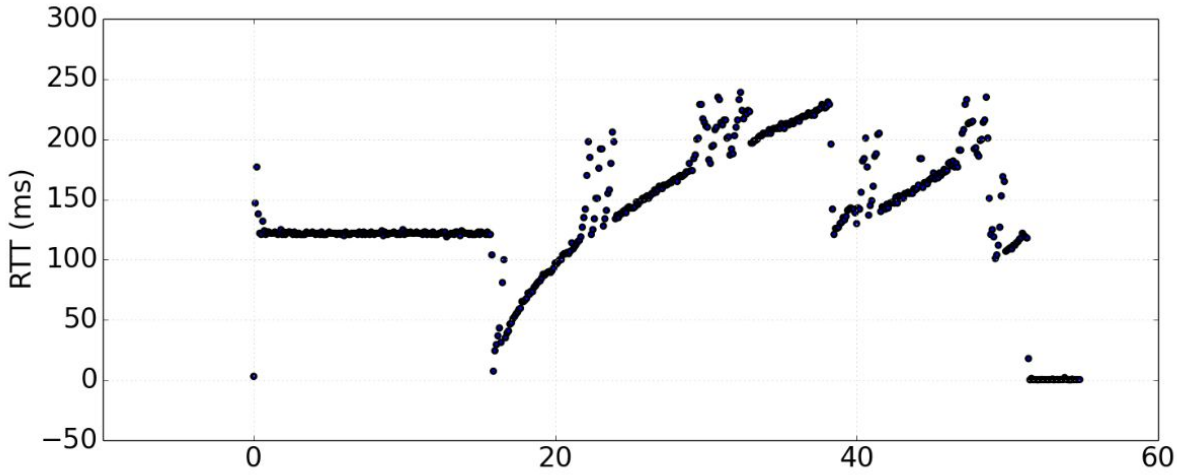
1) *The Long Lived TCP flow cwnd:-*



2) *The RTT reported by ping :-*



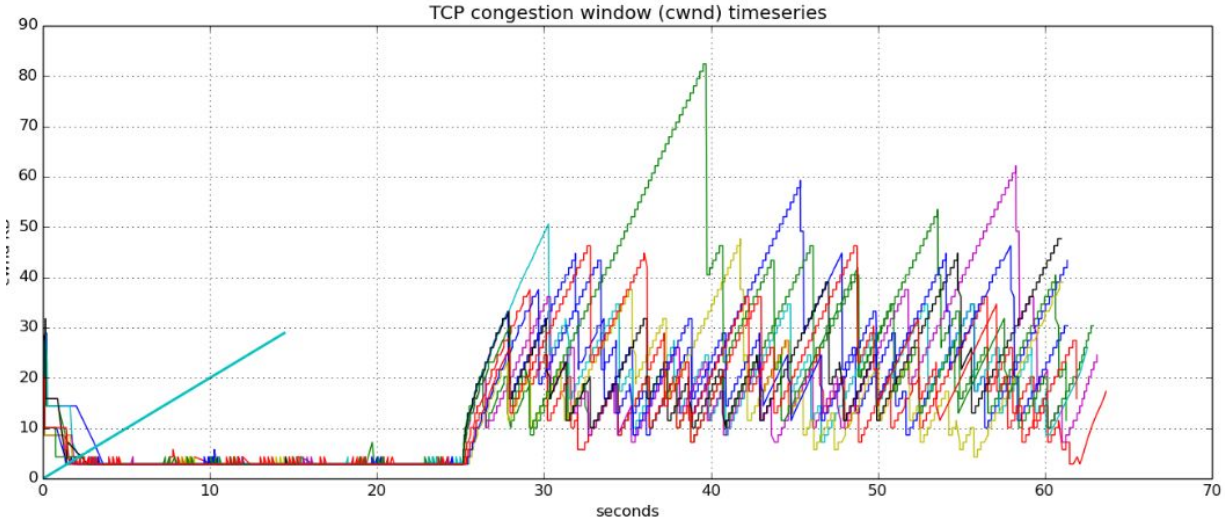
3) *Queue Size at bottleneck :-*



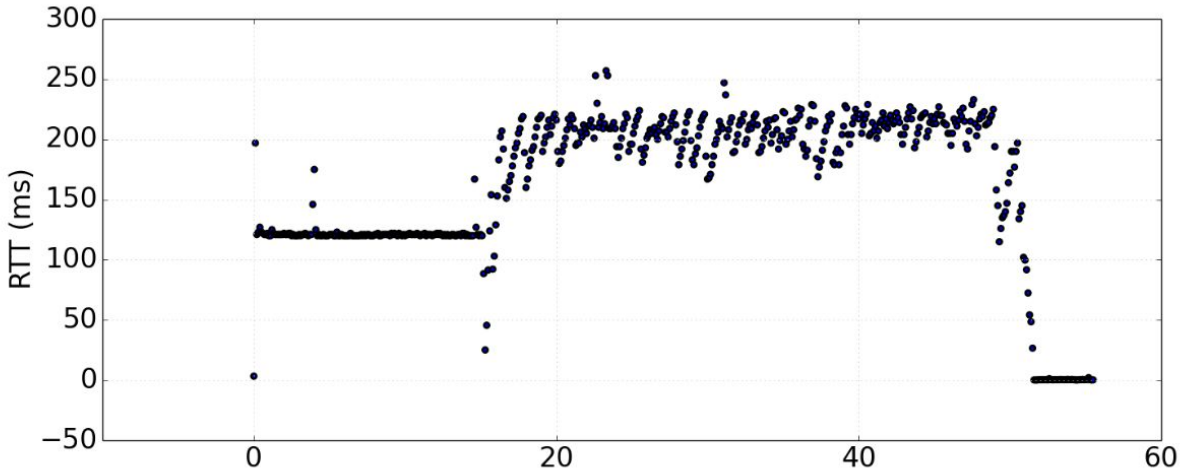
UDP server analysis:-
Jitter = 0.288ms
Loss = 4.83 %
Latency :- 120.51ms

Part3:- AQM PIE Enabled and 10 parallel TCP flow

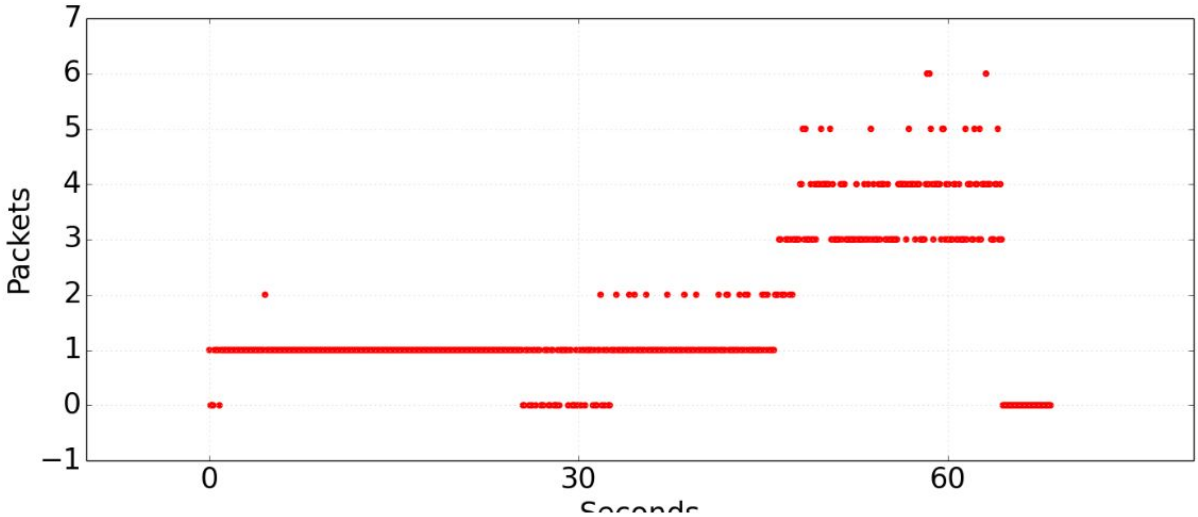
1) *The Long Lived TCP flow cwnd:-*



2) *The RTT reported by ping :-*



3) *Queue Size at bottleneck :-*



UDP server analysis:-
Jitter = 0.279 ms
Loss = 10.3%
Latency :- 22.50 ms