Assignment 1- Bufferbloat Implementation
Advanced Computer Networks- COL724

Zilmarij Iqbal
2019CSY7584

February 3, 2021

1 Bufferbloat implementation

Bufferbloat is the phenomenon that occurs due to TCP congestion. TCP congestion is the consequence of huge bandwidth difference between uplink and downlink across a network.

1.1 Long Lived TCP flows

This can be created using *iperf*. The runner code, $buffer bloat.py$, does the server setup and initiates the monitors, then the client (h1) initiates the flow.

Following are the graphs for cwnd and queue occupancy:

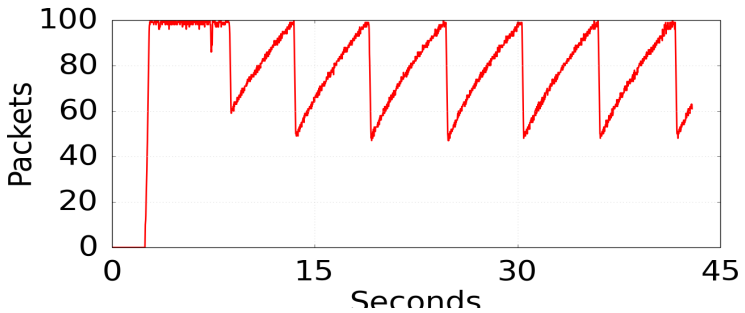

Figure 1: Queue Occupancy at Bottleneck

1.2 Pings from h1 to h2 and the RTT

Command for this:
h1 ping h2 -c 10 -i 0.1 > ping.txt — Using CLI
The observation was that the first ping takes considerably longer than the others. This is because ARP tables, MAC tables, etc, are initialised during the

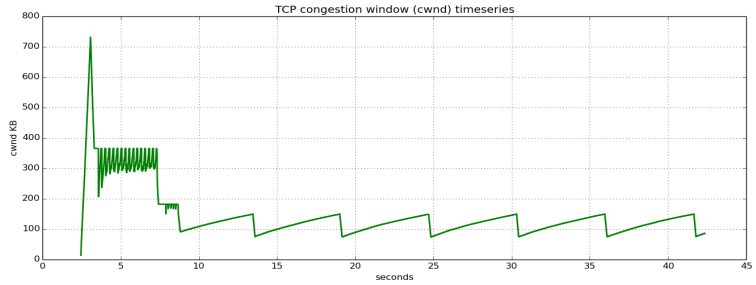

Figure 2: cwnd

first ping. Figure 3 shows the plot for ping stats:

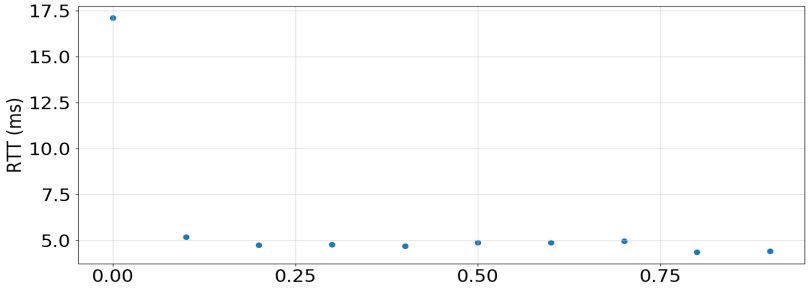

Figure 3: RTT

1.3 Spawning a web server at h2

The average wall clock time for 10 downloads was found to be 10*sec*. Average Download time was found to be 0.06*sec*.

1.4 Bufferbloat with PIE AQM enabled

The file *bb.py* contains the code for setting up the topology with PIE AQM enabled. Rest of the usage remains the same.

When we try to download a webpage with PIE AQM with target delay enabled, we observe that it takes longer as compared to the scenario without PIE AQM. The average downloading time in this case was observed to be 0.3*sec*(while it was observed to be 0.06sec in the original case).

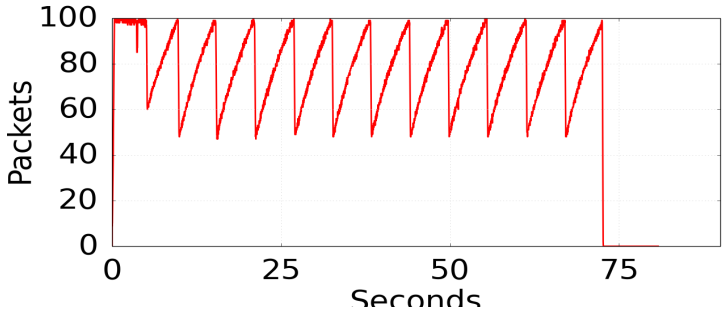

Figure 4: Queue Occupancy at Bottleneck with PIE AQM enabled

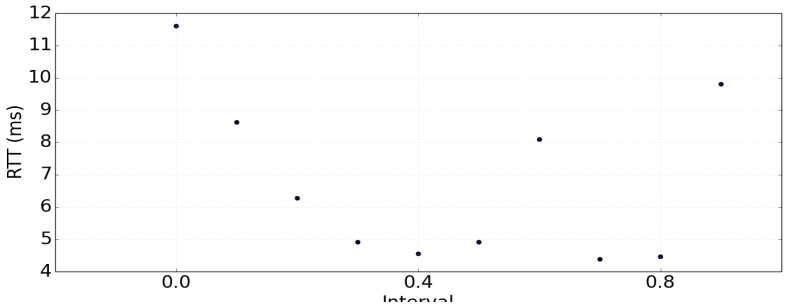

Figure 5: RTT with PIE AQM enabled

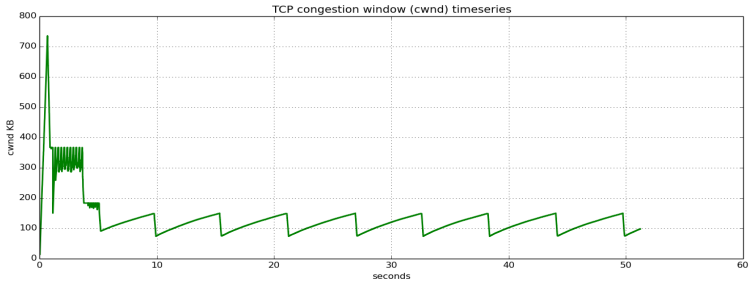

Figure 6: cwnd with PIE AQM enabled

1.5 Bufferbloat with PIE AQM and 10 TCP flows

For generating simultaneous TCP flows, we use the $-P$ flag while initiaitng TCP connection from the client's side using $iperf$. Figures 7,8,9 represent the corresponding graphs for this scenario:

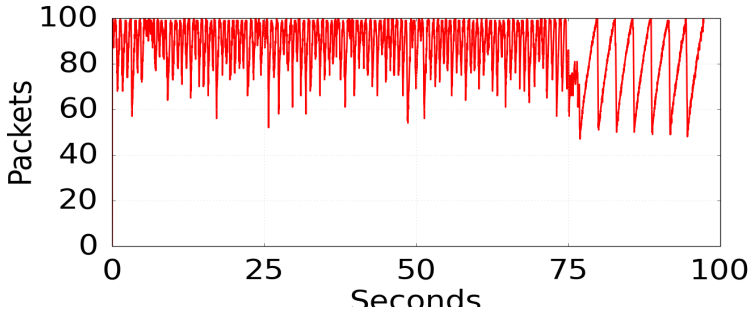

Figure 7: Queue Occupancy at Bottleneck with PIE AQM and 10 TCP flows

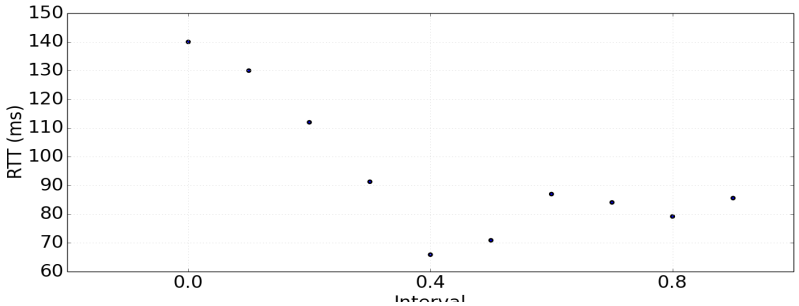

Figure 8: RTT with PIE AQM and 10 TCP flows

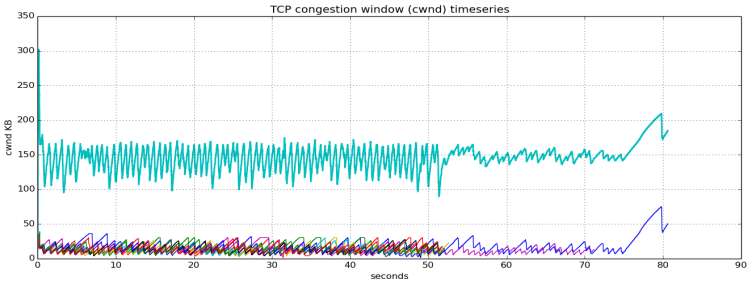

Figure 9: cwnd with PIE AQM and 10 TCP flows

2 Wireless Bufferbloat

Wireless link with certain loss percentage(s) were emulated using the wired link itself. This is done by adding the *loss* field and setting it to the required value. Following are the required graphs for loss percentages of 0.5%, 1%, 2% and 4%, resp.

2.1 Graphs for loss percentage of 0.5%

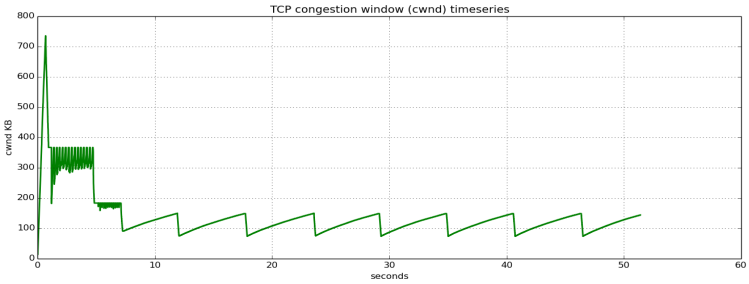

Figure 10: Cwnd with loss of 0.5%

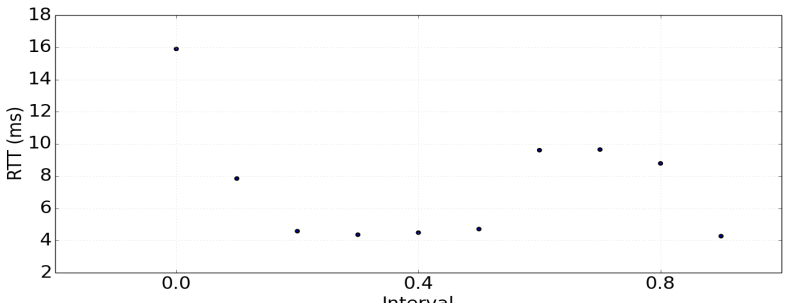

Figure 11: RTT with loss of 0.5%

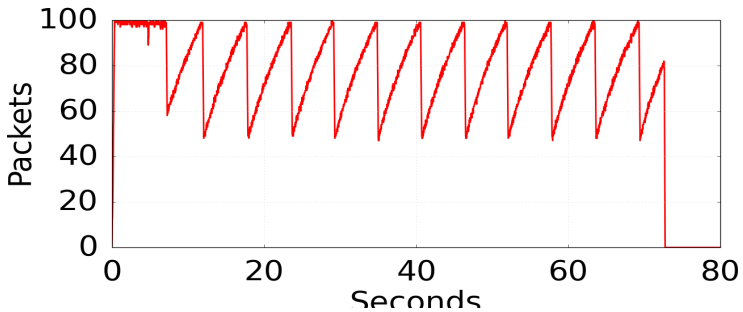

Figure 12: Queue Occupancy at Bottleneck with loss of 0.5%

2.1.1 Graphs for loss percentage of 0.5% with PIE AQM enabled

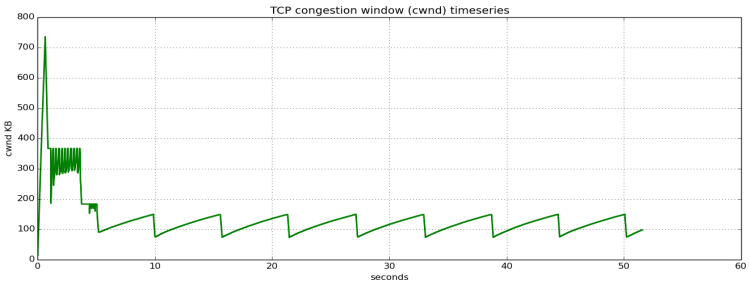

Figure 13: Cwnd with loss of 0.5% with PIE AQM enabled

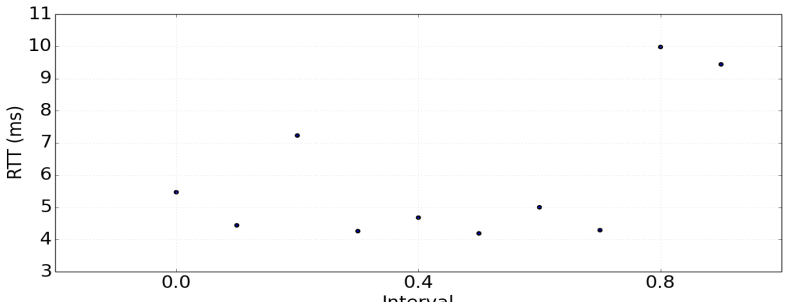

Figure 14: RTT with loss of 0.5% with PIE AQM enabled

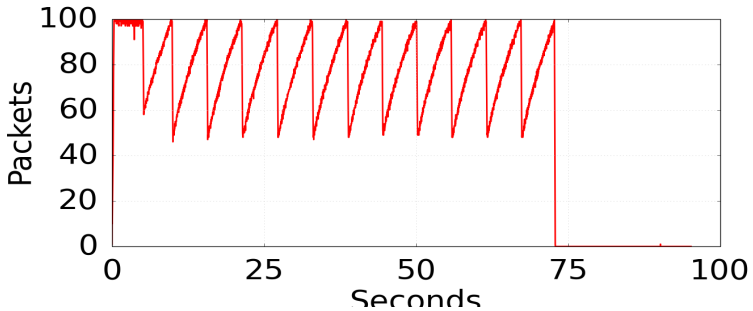

Figure 15: Queue Occupancy at Bottleneck with loss of 0.5% with PIE AQM enabled

2.1.2 Graphs for loss percentage of 0.5% with PIE AQM enabled and 10 TCP flows

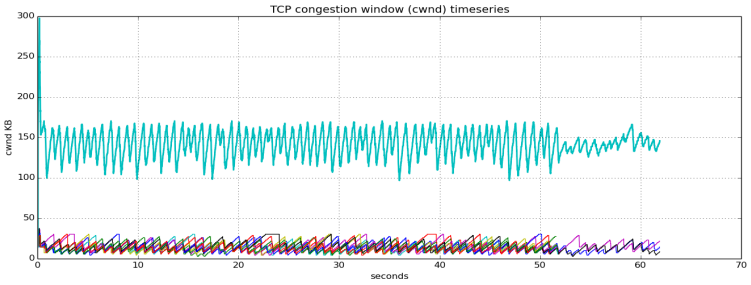

Figure 16: Cwnd with loss of 0.5% with PIE AQM enabled and 10 TCP flows

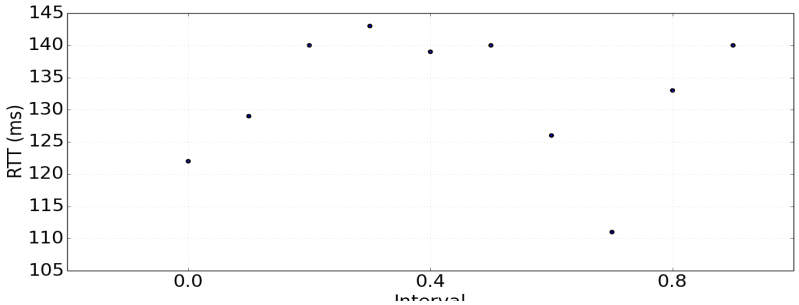

Figure 17: RTT with loss of 0.5% with PIE AQM enabled and 10 TCP flows

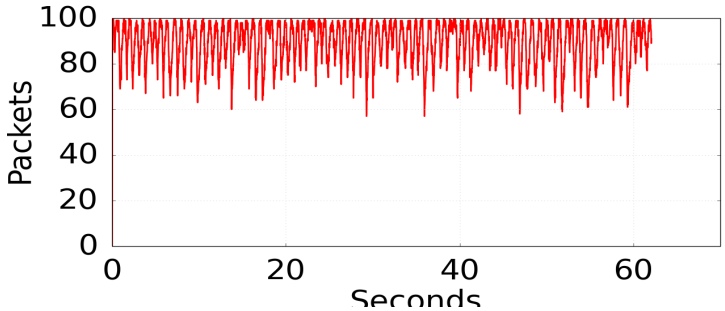

Figure 18: Queue Occupancy at Bottleneck with loss of 0.5% with PIE AQM enabled and 10 TCP flows

2.2 Graphs for loss percentage of 1%

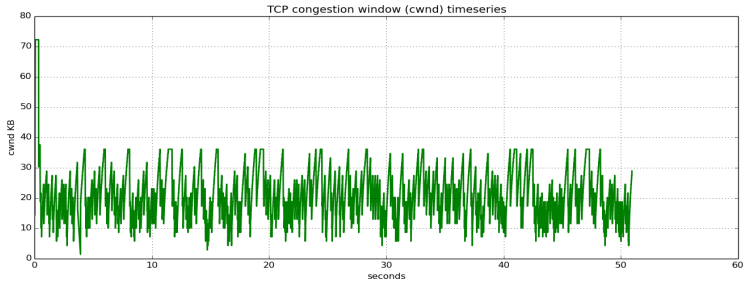

Figure 19: Cwnd with loss of 1%

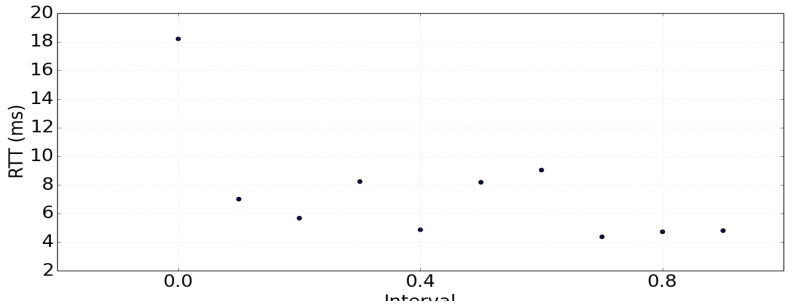

Figure 20: RTT with loss of 1%

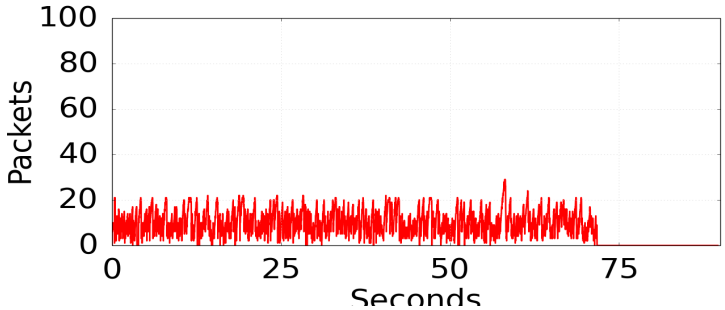

Figure 21: Queue Occupancy at Bottleneck with loss of 1%

2.2.1 Graphs for loss percentage of 1% with PIE AQM enabled

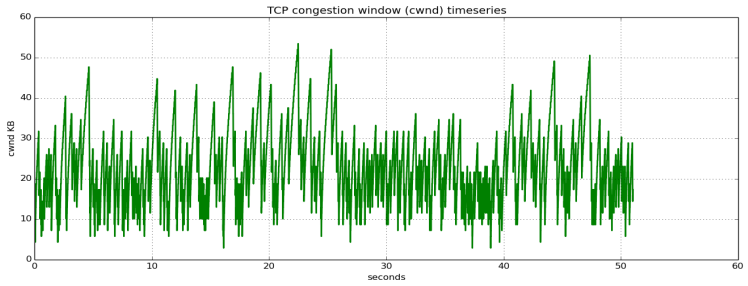

Figure 22: Cwnd with loss of 1% with PIE AQM enabled

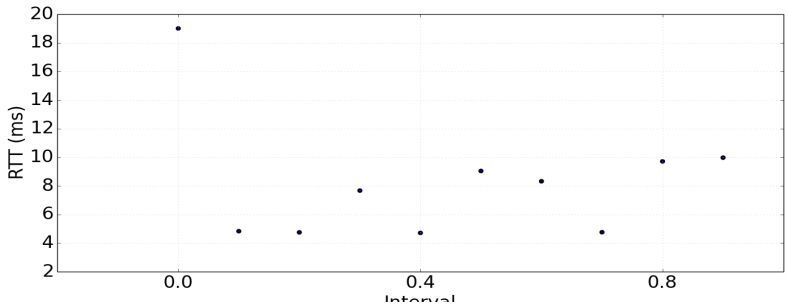

Figure 23: RTT with loss of 1% with PIE AQM enabled

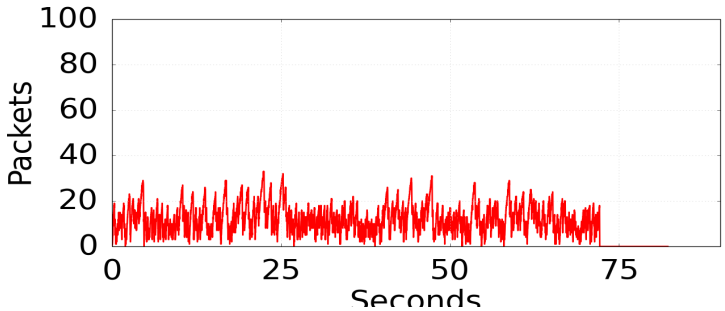

Figure 24: Queue Occupancy at Bottleneck with loss of 1% with PIE AQM enabled

2.2.2 Graphs for loss percentage of 1% with PIE AQM enabled and 10 TCP flows

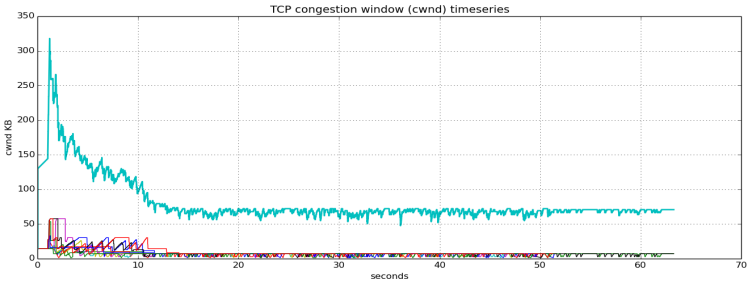

Figure 25: Cwnd with loss of 1% with PIE AQM enabled and 10 TCP flows

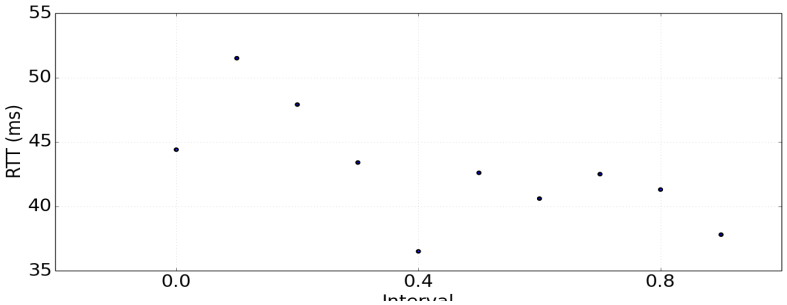

Figure 26: RTT with loss of 1% with PIE AQM enabled and 10 TCP flows

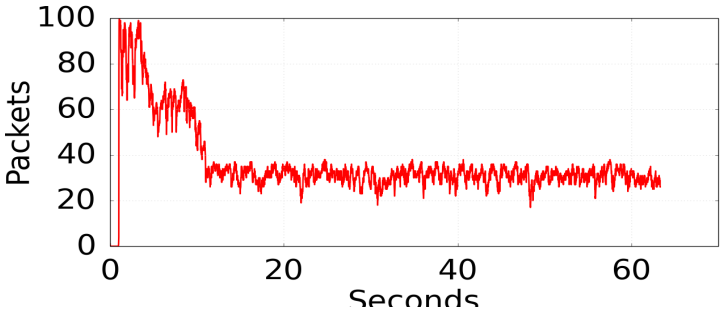

Figure 27: Queue Occupancy at Bottleneck with loss of 1% with PIE AQM enabled and 10 TCP flows

2.3 Graphs for loss percentage of 2%

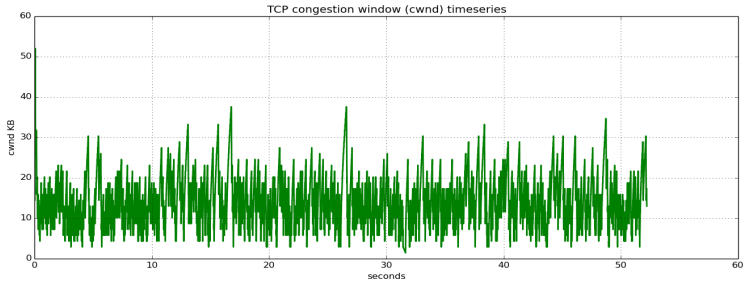

Figure 28: Cwnd with loss of 2%

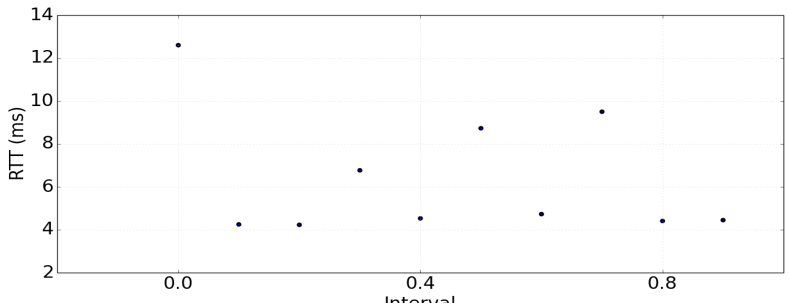

Figure 29: RTT with loss of 2%

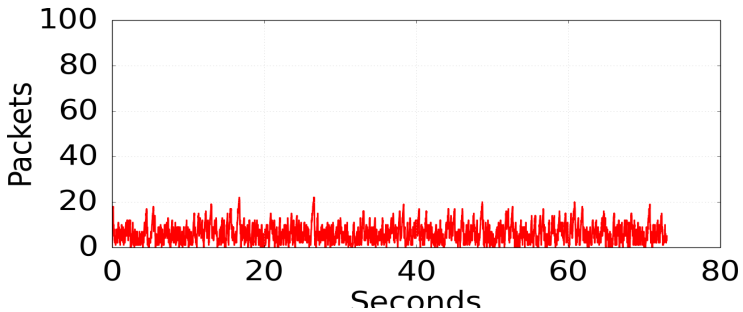

Figure 30: Queue Occupancy at Bottleneck with loss of 2%

2.3.1 Graphs for loss percentage of 2% with PIE AQM enabled

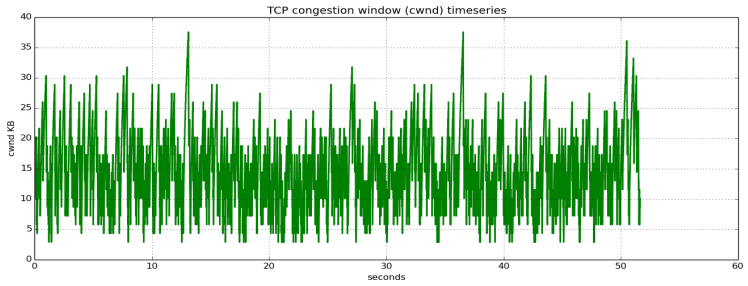

Figure 31: Cwnd with loss of 2% and PIE AQM enabled

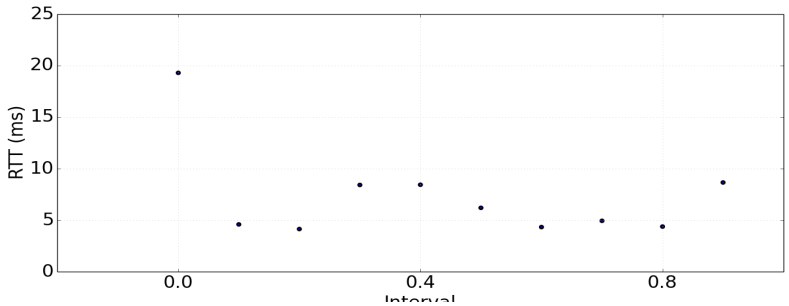

Figure 32: RTT with loss of 2% and PIE AQM enabled

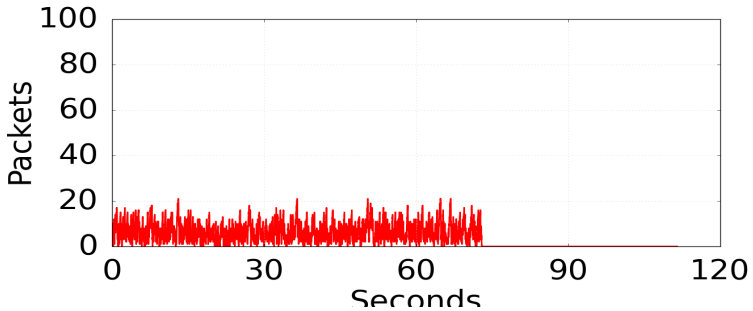

Figure 33: Queue Occupancy at Bottleneck with loss of 2% and PIE AQM enabled

2.3.2 Graphs for loss percentage of 2% and PIE AQM enabled with 10 TCP flows

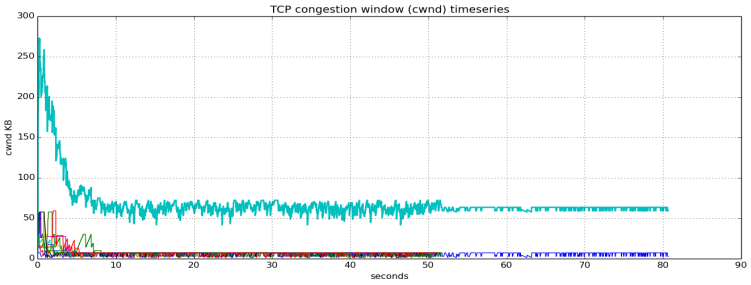

Figure 34: Cwnd with loss of 2% and PIE AQM enabled with 10 TCP flows

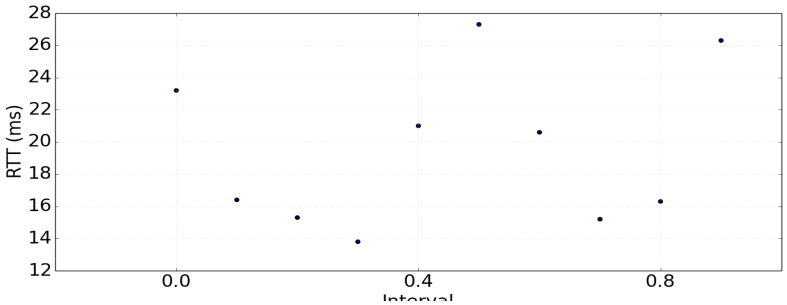

Figure 35: RTT with loss of 2% and PIE AQM enabled with 10 TCP flows

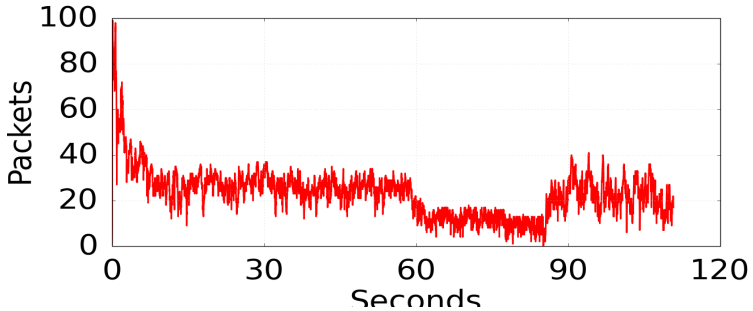

Figure 36: Queue Occupancy at Bottleneck with loss of 2% and PIE AQM enabled with 10 TCP flows

2.4 Graphs for loss percentage of 4%

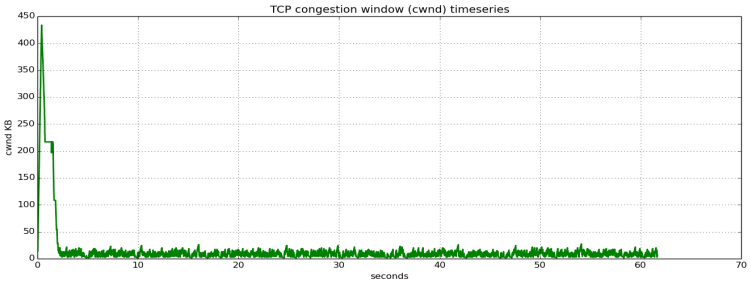

Figure 37: Cwnd with loss of 4%

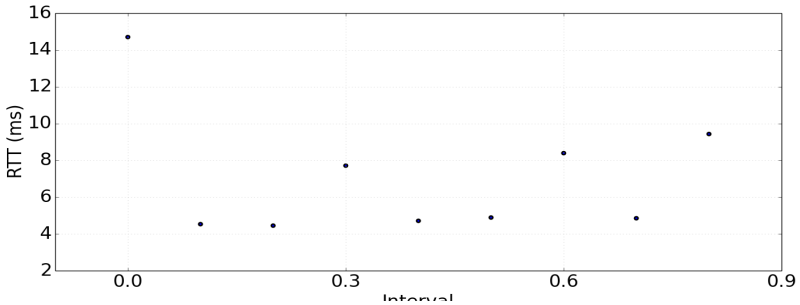

Figure 38: RTT with loss of 4%

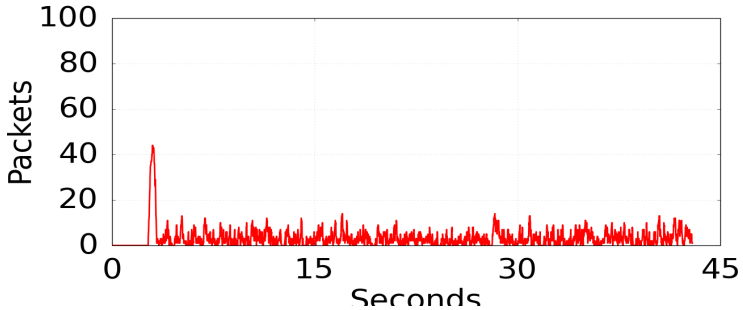

Figure 39: Queue Occupancy at Bottleneck with loss of 4%

2.4.1 Graphs for 4% loss with PIE AQM enabled

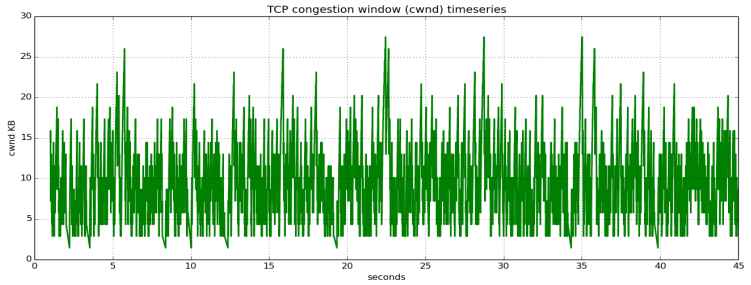

Figure 40: Cwnd with PIE AQM with loss of 4%

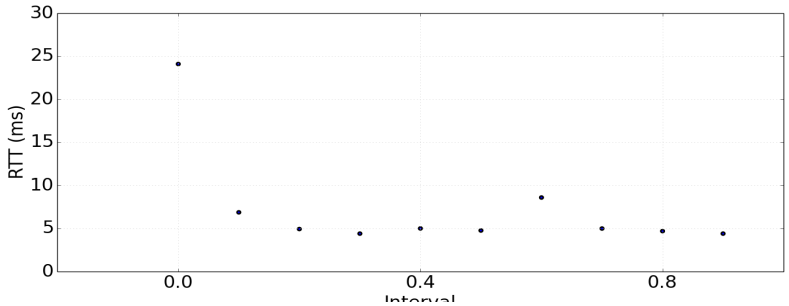

Figure 41: RTT with PIE AQM with loss of 4%

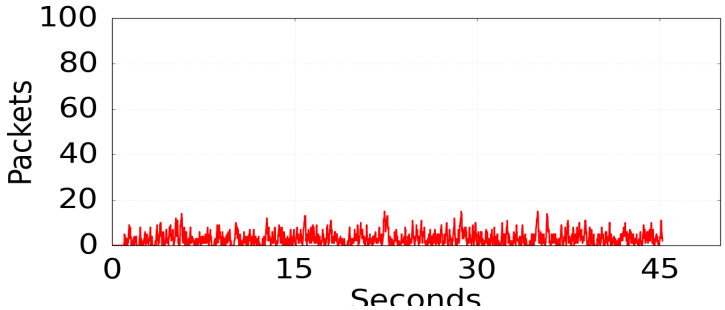

Figure 42: Queue Occupancy at Bottleneck with PIE AQM with loss of 4%

2.4.2 Graphs for 4% loss with 10 TCP flows and PIE AQM enabled

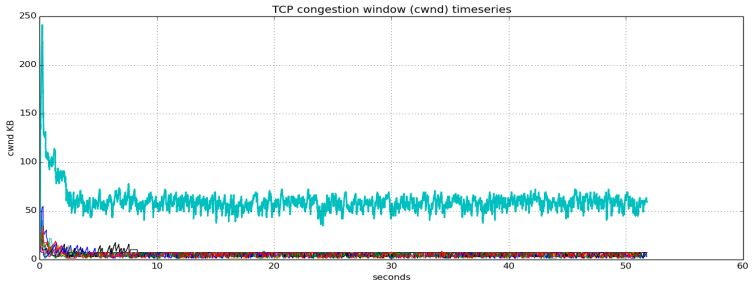

Figure 43: Cwnd with PIE AQM and 10 TCP flows, with loss of 4%

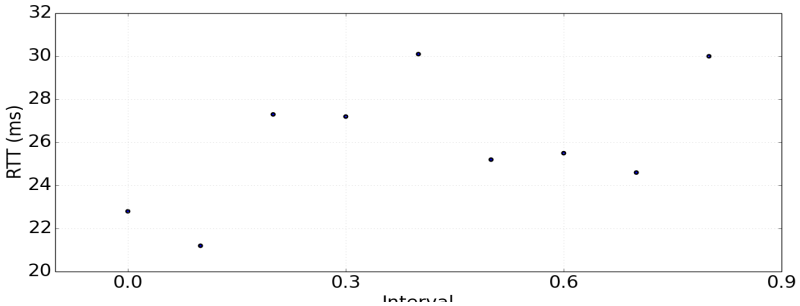

Figure 44: RTT with PIE AQM and 10 TCP flows, with loss of 4%

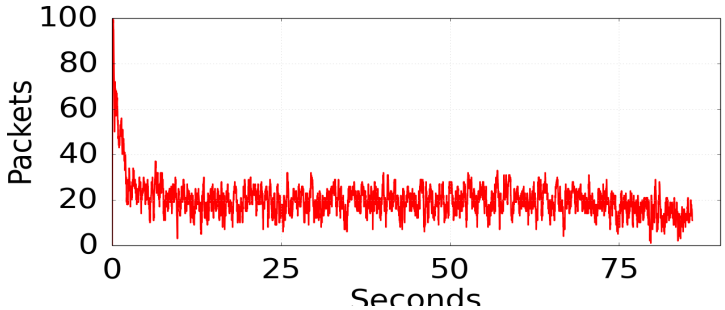

Figure 45: Queue Occupancy at Bottleneck with PIE AQM and 10 TCP flows, with loss of 4%

3 TCP and UDP

In this experiment, video flow is emulated by a long-lived, high data rate stream (in UDP). Concurrently, shorter-lived TCP flow emulates packet exchange:

h2.cmd('iperf -s -w 16m -p 5001 -i 1 > iperf-recv_s.txt &')
h2.cmd('iperf -s -u -w 16m -p 5002 -i 1 > iperf-recv_s2.txt &')

start_tcpprobe()
qmon=start_qmon()
h1.cmd('iperf -c 10.0.0.2 -u -b 16Mb -p 5002 -t 50 -w 16m > iperf_recv_c2.txt &')
h1.cmd('iperf -c 10.0.0.2 -p 5001 -t 17 -w 16m > iperf_recv_c.txt &')

3.1 Graphs for TCP

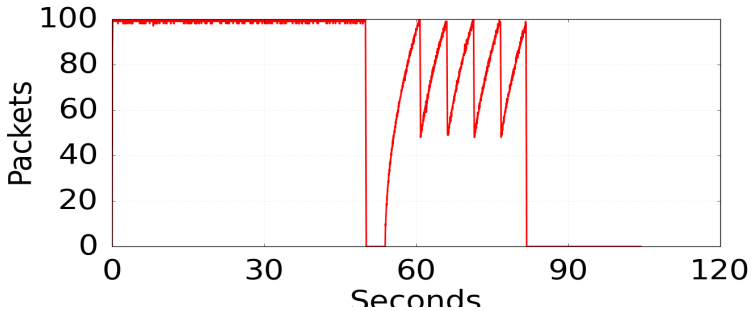

Figure 46: Queue Occupancy at Bottleneck with UDP flow rate of 10MB

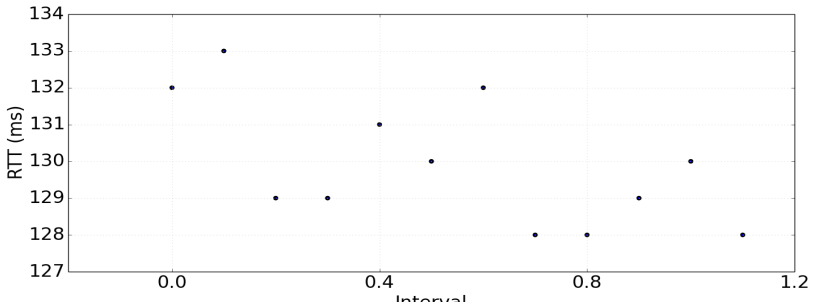

Figure 47: Ping stats with UDP flow rate of 10MB

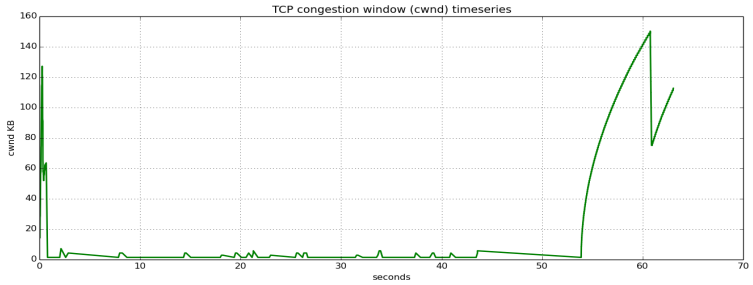

Figure 48: Cwnd with UDP flow rate of 10MB

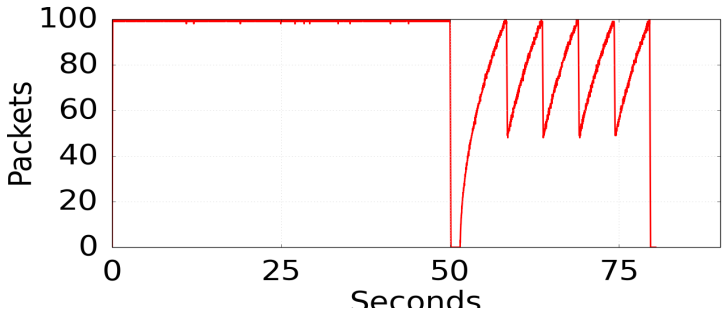

Figure 49: Queue Occupancy at Bottleneck with UDP flow rate of 16MB

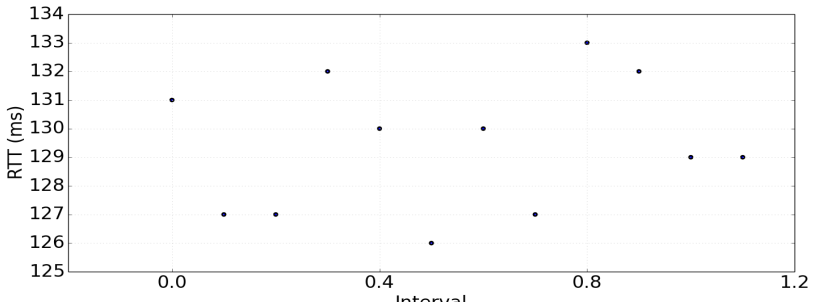

Figure 50: Ping stats with UDP flow rate of 16MB

3.2 Stats for UDP

3.2.1 For Data Rate of 10MB

1. JITTER: 7.799ms
2. LOSS: 16%, 7356/44603

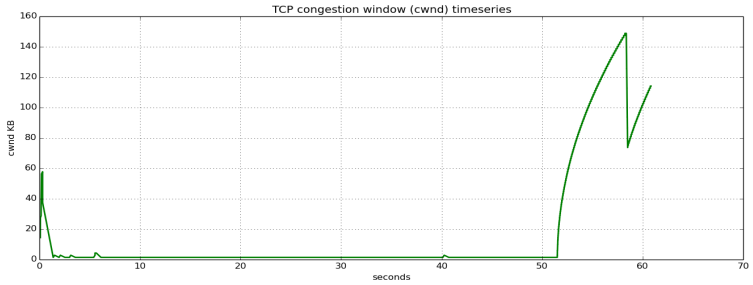

Figure 51: Queue Occupancy at Bottleneck with UDP flow rate of 16MB

3.2.2 For Data Rate of 16MB

1. JITTER: 1.442ms
2. LOSS: 45%, 32449/71427

3.2.3 For default setting

1. JITTER: 1.121ms
2. LOSS: 1.7%

4 Observations

Losses due to noisy links cause a reduction in the link occupancy (and congestion window) as well.
The PIE AQM managing the traffic at the bottleneck interface of the switch causes the queue occupancy to dip (esp. the peak queue occupancy). Also, the target delay of $20ms$ incurs its cost on communication/data transmission, and the effect can be clearly observed through web-page download time.
UDP stream with higher data rate suffers more data loss (UDP is unreliable and does not involve re-transmission of lost packets). Lost packets do not cause any more burden on the network (and on the cwnd, consequently), and any retransmissions that take place are for the TCP flow only (which is subtle as compared to the concurrent UDP flow). On account of this, the TCP flow's congestion window is observed to be having a low-lying graph. Queue occupancy, on the other hand, is a function of *all* the data flowing through the network, and therefore, the queue occupancy reaches its peak when high rate data from UDP and standard rate data TCP both flow simultaneously.