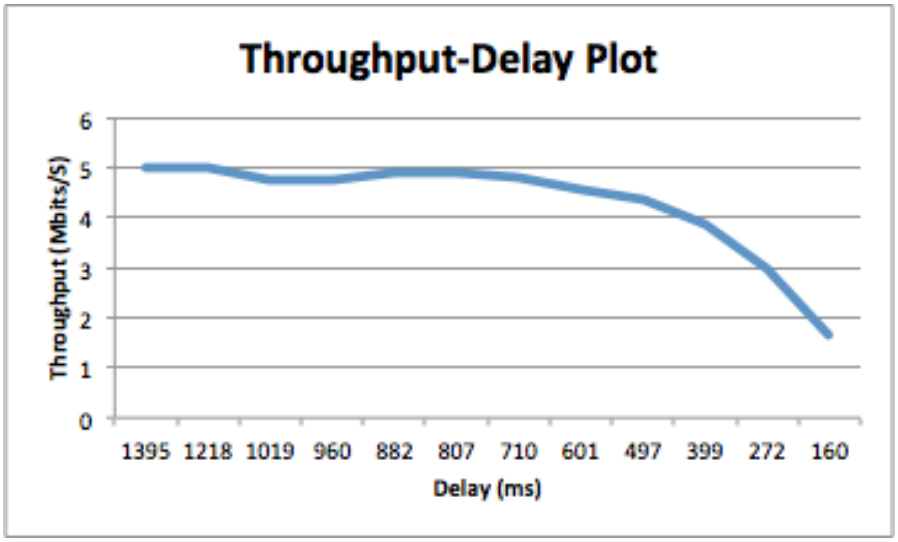
Tiffany Le
Amanda Ke

6.829 Pset 2 Write Up

A)

Window Size	Throughput	95-Percentile Signal Delay	Score: (throughput/delay)
10	1.64 Mbits/s	160 ms	10.25
20	2.98 Mbits/s	272 ms	10.95
30	3.88 Mbits/s	399 ms	9.72
40	4.36 Mbits/s	497 ms	8.77
50	4.54 Mbits/s	601 ms	7.55
60	4.81 Mbits/s	710 ms	6.77
70	4.91 Mbits/s	807 ms	6.08
80	4.91 Mbits/s	882 ms	5.57
90	4.77 Mbits/s	960 ms	4.97
100	4.77 Mbits/s	1019 ms	4.68
125	5.01 Mbits/s	1218 ms	4.11
150	4.98 Mbits/s	1395 ms	3.56

Graph of Throughput vs Delay:



The best window size was around 20 packets. The measurements over multiple runs was fairly consistent and repeatable, with only slight variations in delay and throughput across runs with the same window size.

B) The AIMD Scheme worked moderately well. The throughput was 4.76 Mbits/s and the 95 percentile signal delay was 873 ms. This provides a score of 5.33. While the utilization of the network was fairly high, the signal delay could definitely be reduced. The constants we chose were an additive increase of 1/current_window_size and a multiplicative decrease of 2. Increasing by 1/current_window_size on each successful ACK will linearly increase the congestion window in each RTT. Decreasing the window size by a factor of 2 on congestion also provides adequate room for recovery.

C)
Round trip time (RTT) was calculated by taking the difference between the time when a packet was sent and when an ack for that packet was received. If the RTT was below a threshold, the window size was increased and the threshold was decreased. If the RTT was above a threshold, the window size was decreased and the threshold was increased.
This scheme did not work as well the AIMD scheme; however if we had more time to play with the parameters in the protocol, this scheme could probably have performed better.

Window	Threshold	Power
Additive increase, multiplicative decrease (2)	constant subtractive decrease(10), multiplicative increase (2)	4.71

Constant additive increase(1), multiplicative decrease (2)	constant subtractive decrease(10), multiplicative increase(2)	4.643
Constant additive increase(1), multiplicative decrease (2)	constant subtractive decrease (10), constant multiplicative increase (100)	8.610

D)

Our protocol first estimates the bandwidth, by keeping track of the difference in the number of sent and received packets in the previous time interval. We also use an estimation of the round trip time (RTT), an average of the difference between the sent timestamp and the acknowledgement received timestamp and the RTT in the previous timestep in our bandwidth estimation.

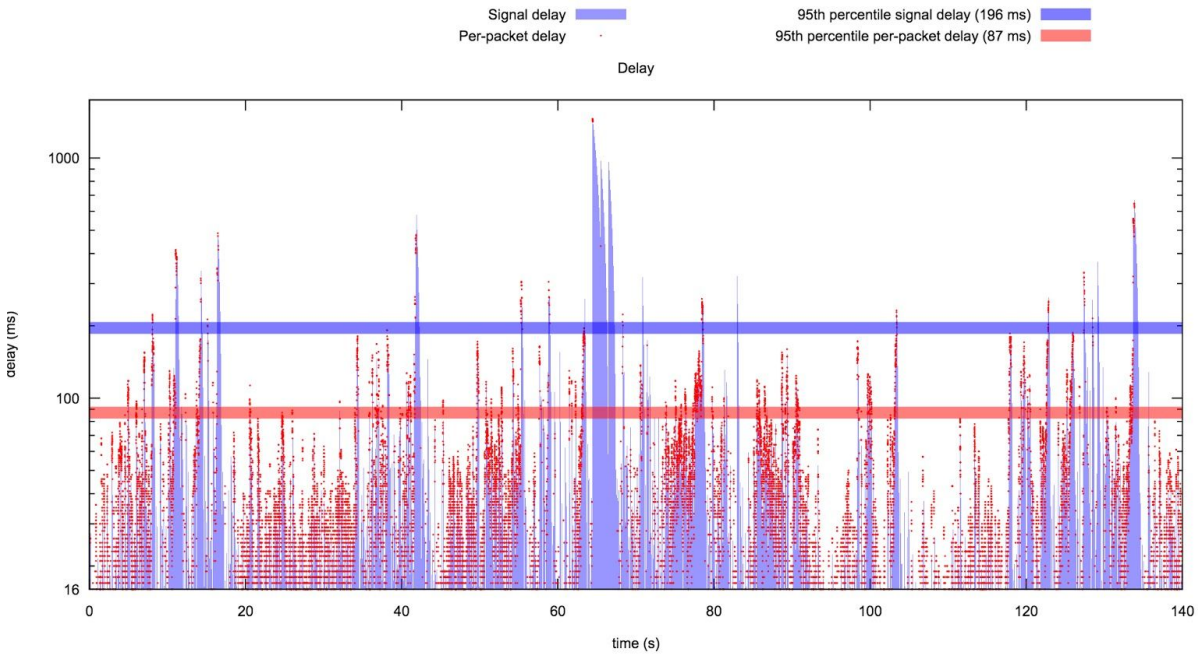
It then calculates the amount of packets the queue can hold before the signal delay is higher than our target signal delay (which we set to 100ms). This capacity was equal to target latency * bandwidth.

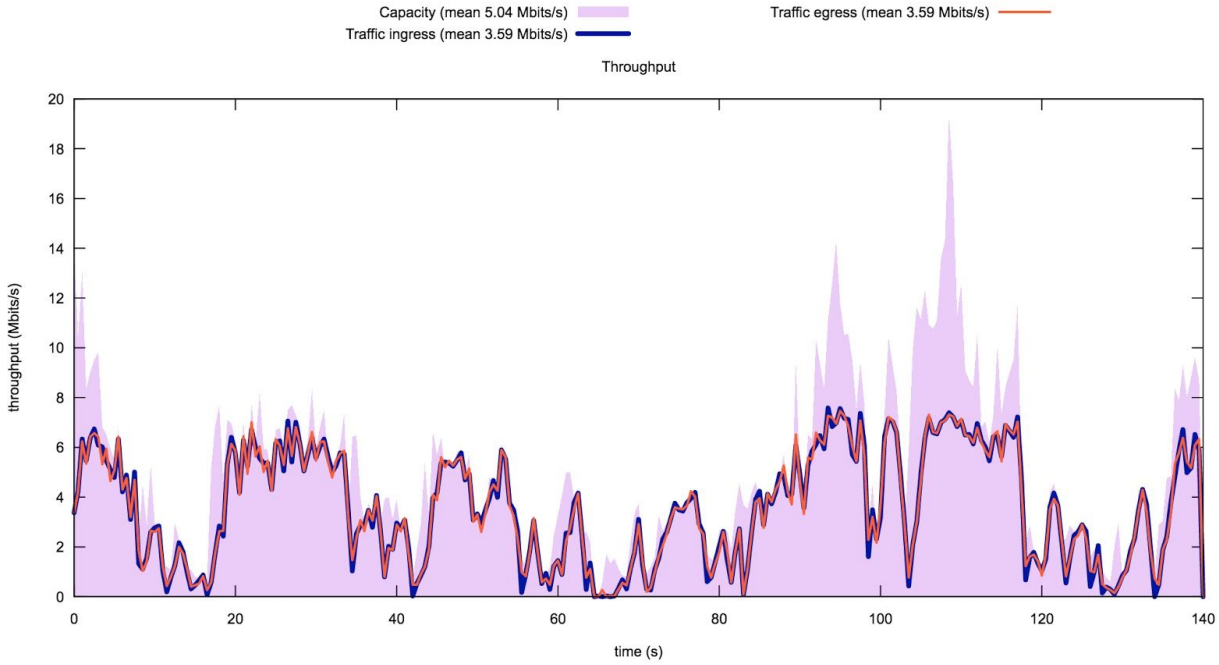
Afterward, it then calculates the number of packets that the network bottleneck can process, and adds that number to the capacity. This number was calculated as RTT/2*bandwidth.

If the capacity is larger than the current window size, that means the network can handle more packets without causing significant delays, so window size is increased by 1/window size. If the capacity is smaller than the current window size, the window size is decreased by 5 packets.

To retain valid window sizes, we kept a min cap of 1 packet, and a max cap of 35 packets.

WIth the Verizon 140 second trace, our throughput was 3.59 Mbits/sec and our 95th percentile signal delay was 196 ms. This gave us a score of 18.32. The plots below showcase the delay and throughput when running the trace.


Signal delay
Per-packet delay
95th percentile signal delay (196 ms)
95th percentile per-packet delay (87 ms)
Delay
1000
100
16
delay (ms)
0
20
40
60
80
100
120
140
time (s)


Capacity (mean 5.04 Mbits/s)
Traffic ingress (mean 3.59 Mbits/s)
Traffic egress (mean 3.59 Mbits/s)
Throughput
20
18
16
14
12
10
8
6
4
2
0
throughput (Mbits/s)
0
20
40
60
80
100
120
140
time (s)