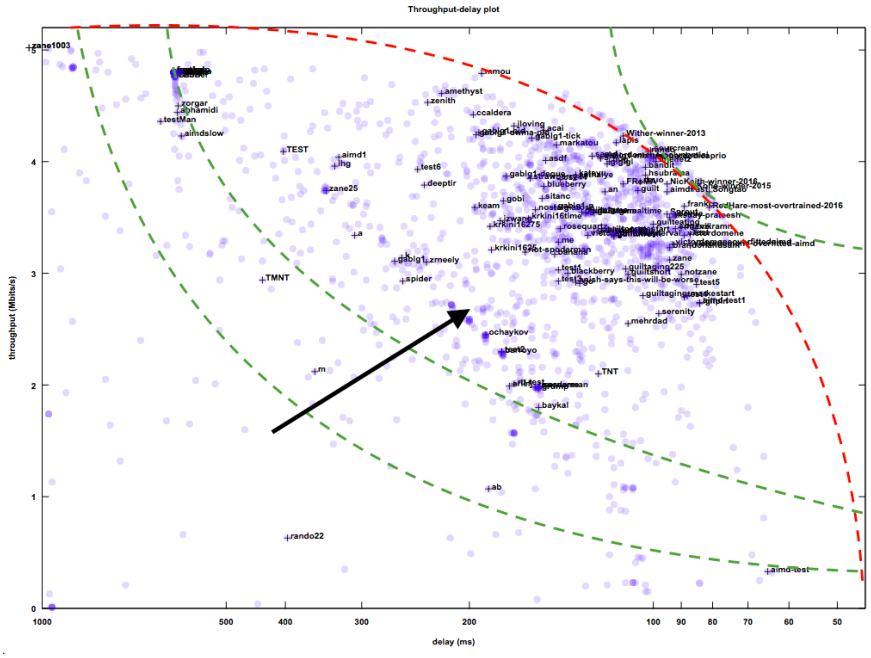
6.829 PSet2

Mehrdad Khani

Exercise A:

This is a typical figure frequently appearing in bandwidth-delay discussions which means you can not go above an specific throughput for each delay on the horizontal line. The throughput axis boundary is the link capacity and the delay axis is bounded to minimum Round Trip Time. The best constant window size measurement would be the one a the intersection of the drawn red and green curves. The green lines are power contours which increase in the direction of the arrow in the figure. In terms of being repeatable, the experiments are fairly repeatable. The details of what happen during the experiment to your packets at each time step can be different. The performance of the simulation is dependent on the hardware machine it is being implemented on and it can vary depending on the number of cores your machine have, the workloads,..

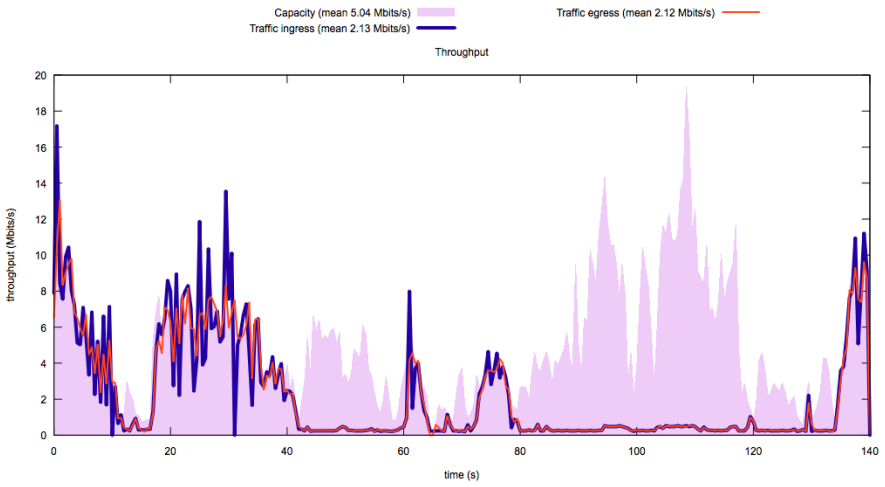


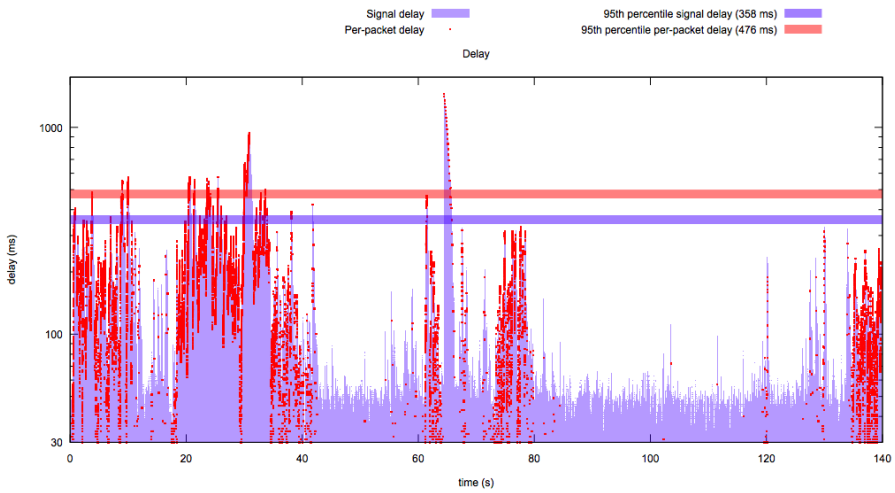
Exercise B:

In this section we have implemented an AIMD (Additive Increase Multiplicative Decrease) congestion control algorithm. AIMD has its own pros among which stand robustness and ease of implementation. Also it does not need to set many parameter, but if you are to going to make it optimized for a special purpose (delay, throughput, power, ..) it will not be an easy job. Below you can find what I got for implementing it on the Verizon data with $A = 1$ and $M = 0.5$. It worth mentioning that it is possible to come up with even better results by tweaking these two parameters in addition to the timeout period for not getting an ack which are set manually in this algorithm. One other interesting property of this assignment is the sensitivity of algorithms to the resending timeout which has lied wickedly at the last line of controller.cc and is responsible for avoiding getting out of control signals in these scenarios. As you know, the sender needs to send at least some packets sometimes to the receiver to get aware of the link and buffers status. Altogether, AIMD is pretty good in terms of robustness, it will converge even with bad choices of initial values or parameters for its dynamic properties.

Summary statistics

```
Average capacity: 5.04 Mbits/s
Average throughput: 2.12 Mbits/s (42.1% utilization)
95th percentile per-packet queueing delay: 476 ms
95th percentile signal delay: 358 ms
```





Exercise C:

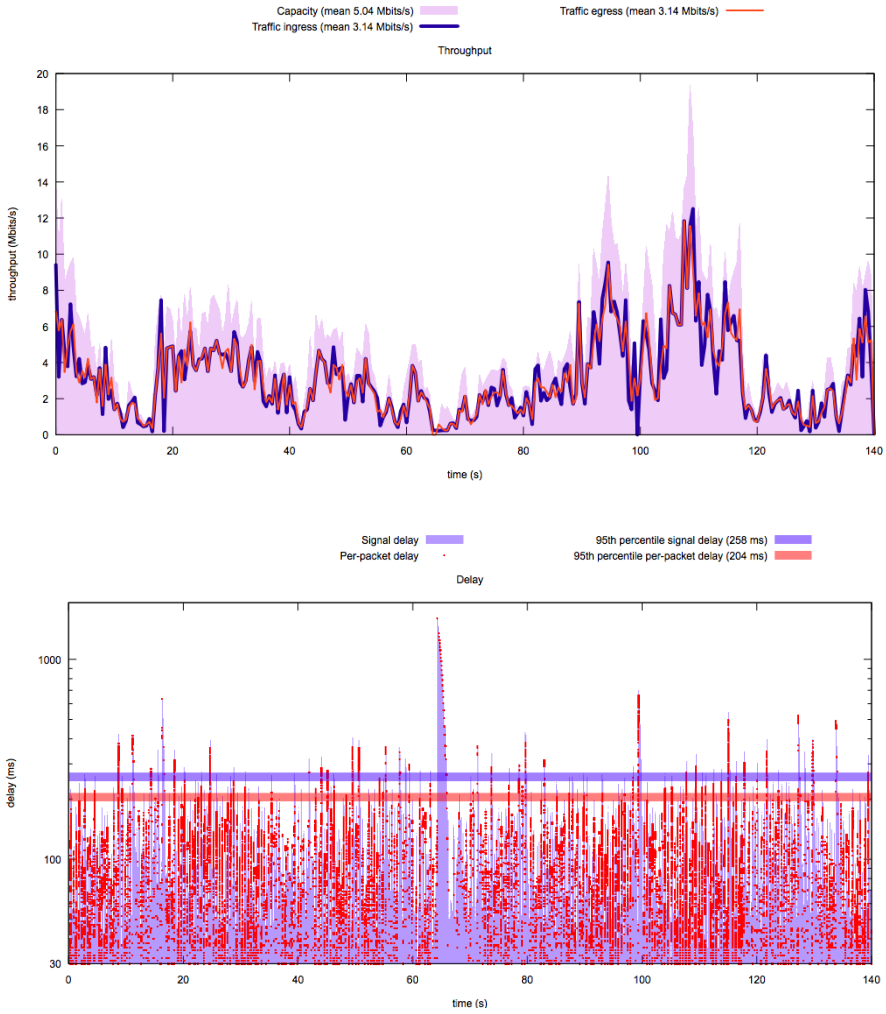
Delay-triggered schemes can be tricky. One should care about the initial values he picks and you should set as much parameters as you need to achieve the desired behavior which is prone to overfitting for an special case or dataset. In this part, I used two thresholds in order to obtain an acceptable performance of this delay triggered algorithm alone:

- RTT $\leq$ *th1*: fast window size increment
- *th1* $\leq$ RTT $<$ *th2*: slow window size decrease
- *th2* $\leq$ RTT: fast window size decrease

The performance of this algorithm with $th1 = 80$ and $th2 = 110$ is illustrated in the figure below.

Summary statistics

```
Average capacity: 5.04 Mbits/s
Average throughput: 3.14 Mbits/s (62.2% utilization)
95th percentile per-packet queueing delay: 204 ms
95th percentile signal delay: 258 ms
```



One important thing regarding delay-triggered traffic controlling is a lack of bandwidth information. The RTTs do not represent the sizes of queues easily, although there are some possible solutions in order to get an estimate of bandwidth using time signals available which we will explain and use in the next part.

Exercise D:

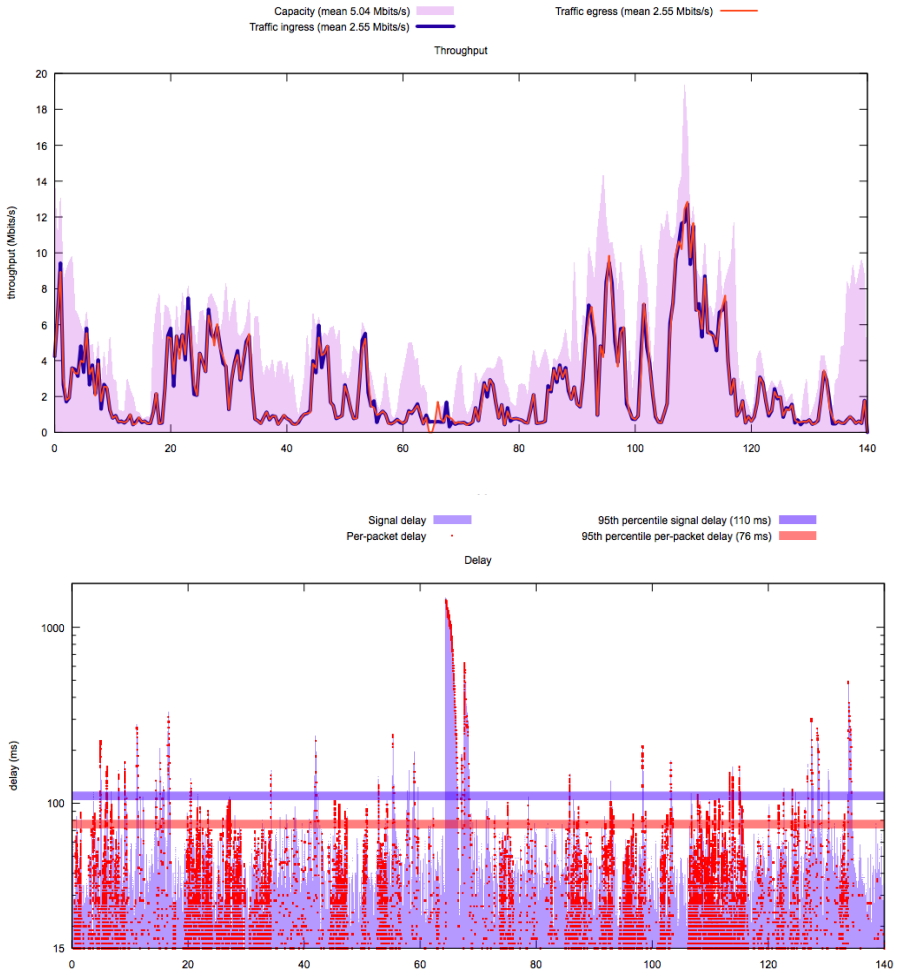
After trying several approaches, I find the following makes most sense:

Use the two control signals: RTT and the delay difference between consecutive ack arrival and their transmitting timestamps. In other words, each two

sequential acks arrived at the sender have experienced the same queuing delays and bandwidth. If we are receiving packets are receiver side at a lower rate, it is then the channel capacity has been the bottleneck and the rate at which we are receiving will be the channel capacity. In opposite case, when are receiving packets faster that how we are putting them on the channel, then we should probably increase or sender rate to achieve the maximum utilization of bandwidth and can calculate the channel bandwidth there again. Thus comparing ack arrivals and corresponding send timestamps will help us figure out the bandwidth. These two signals seem to be very strong in terms of control.

Also there is a need for averaging both signals introduced latter over the time. These two signals are too much changing and make our control problem challenging in case of working with their realtime values. Instead, I took an average over 40ms from ack arrival differences and over 10ms for RTTs. RTTs should have more realtime information I feel, but the channel capacity is changing in a roughly more correlated manner (although it sometimes suddenly fades.) In order to make sure of not getting into long RTTs trouble and lossing control over the system, there is put a multiplicative decrease above the window size which is triggered by the RTT.

Using these two signals, there is implemented an algorithm which might be more robust to overfitting problems and has less parameters to set. In fact, you should mostly decide on bandwidth-delay tradeoff by choosing an RTT threshold. Below you can see one of this algorithm results.


Capacity (mean 5.04 Mbits/s)
Traffic ingress (mean 2.55 Mbits/s)
Traffic egress (mean 2.55 Mbits/s)
Throughput
throughput (Mbits/s)
0
2
4
6
8
10
12
14
16
18
20
0
20
40
60
80
100
120
140
Signal delay
Per-packet delay
95th percentile signal delay (110 ms)
95th percentile per-packet delay (76 ms)
Delay
delay (ms)
15
100
1000
0
20
40
60
80
100
120
140