6.829: Problem Set 2

Sitan Chen, Qinxuan Pan

October 25, 2016

1 Warmup A

We varied the window size from 5 to 50 in increments of five, with two experimental repetitions per value. We found that of these settings, a window size of 5 gave the best ratio of throughput to signal delay (11.42 and 12.05 in the two trials, from throughputs of 2.41 and 2.53 mbps and signal delays of 211 and 210 respectively). The results showed good repeatability, as the average throughput and signal delay varied minimally. The plot of our results is shown in Figure **??**, with the x-axis on a log-scale.

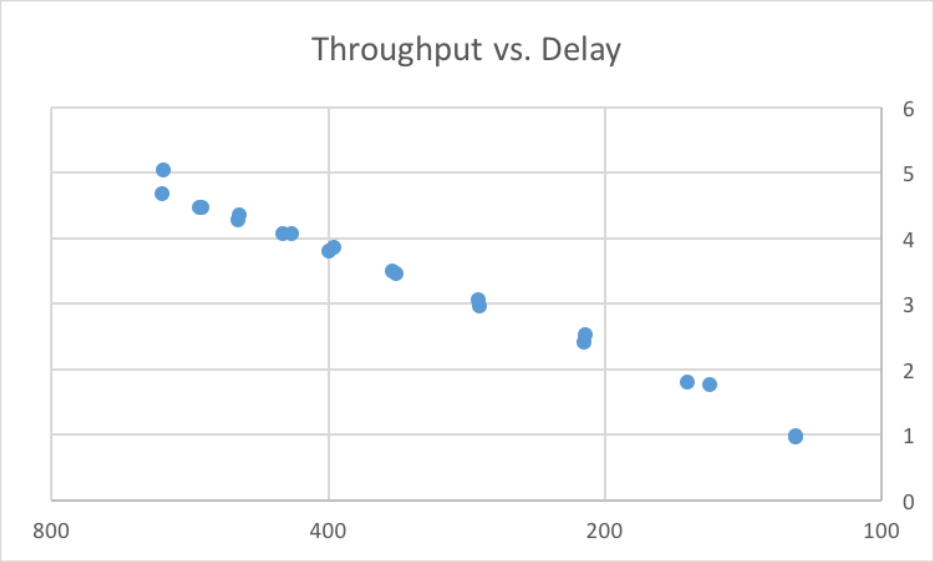

Figure 1: Average throughput (y-axis) vs. signal delay (x-axis)

2 Warmup B

We implemented a simple AIMD scheme where the congestion signal was taken to be when the difference between the timestamp of the current ack that was received and that of the most recent ack exceeds the timeout value. This worked very poorly because acks were received too frequently for multiplicative decreases upon congestion to happen at a reasonable rate; as a result, timeouts made for a very poor signal of congestion.

We initially tried additive increases of 1/cwnd and multiplicative decreases by a factor of 0.5, over the default timeout period of 1000ms, only to discover quickly that window size essentially

increased without bound, leading to traffic overshooting capacity and therefore massive delays. We kept the same AIMD constants and lowered the timeout period of 200ms, yielding throughput of 4.93 mbps and average signal delay of 1406ms. The window size still went as high as 230 with multiplicative decreases happening very rarely; delays went as high as 2 seconds, even though utilization was obviously quite good. We then lowered the timeout period further to 100ms; in this regime window size went up to the 160s with multiplicative decreases happening with more reliable frequency as expected, yielding throughput of 4.65 mbps and a more tolerable average delay of 701 ms.

We reasoned that because multiplicative decreases happen so rarely, it would matter more how quickly/slowly we did additive increases than how aggressively we multiplicatively decreased upon congestion. So the last setting we checked was with AI at 0.5/cwnd, MD at 0.5, and timeout at 100 ms. Window size went as high as 122 with timeouts happening with reasonable frequency, yielding throughput of 3.96 mbps and average delay of 461 ms.

3 Warmup C

In the delay-triggered scheme we implemented, upon the sender's receipt of every ack, we compared EWMA estimates of RTT to a particular delay threshold, treating instances where the former exceeded the latter as signals of congestion. This was built over the same AIMD scheme as in the previous warmup exercise. Overall, we found this scheme to be appreciably more effective, with MD's happening quite frequently; the implication seems that under the highly variable packet delivery rates of cellular networks, delays are a much more informative congestion signal than timeouts.

We kept the AIMD constants fixed at $AI = 1$ and $MD = 0.5$ for simplicity, instead opting to tweak the EWMA weight α and delay threshold T. For $(\alpha, T) = (0.7, 200)$, we found multiplicative decreases to be very frequent, much more so than in Warmup B, with throughput of 3.92 mbps and average signal delay of 172. We then reduced the threshold to $T = 100$ ms; naturally, MD's became even more frequent, with window size only ever going up to the 70s at most, and average signal delay improved at the cost of utilization. In this regime, the throughput was 2.40 mbps and average signal delay was 102 ms. Finally, we reduced α to 0.5, i.e. discounting the past less quickly. The primary qualitative change here was that throughput responded less aggressively to sudden fluctuations in capacity. The net effect was a throughput of 2.57 mbps and average signal delay of 100 ms.

4 Exercise D

Our controller design for the competition consists of two parts. The first part estimates the current available link rate, and the second part estimates (conservatively) the amount of data that will go through within the next 100ms, based on the estimated link rate. Here, 100ms can be thought of as our target delay. The result from the second part is used to set the congestion window size, because that is the amount of data we can put out now that we believe will be processed within the targeted delay. We run our estimators once per tick, which is set to 20ms.

The first part is very simple. We simply maintain an EWMA average for the link rate λ:

$$\lambda = (1 - \alpha)\lambda + \alpha\lambda_{now}.$$

Here, λ_{now} is the measured rate within the last tick. We use an α value of 0.7, because the widely fluctuating nature of wireless channels means that we can disregard the past quickly.

We choose to maintain a single value as our estimate for link rate, instead of a distribution on many possible link rate values (as in Sprout), again due to our belief that the current link rate is best inferred from its measured performance during the most recent several ticks (which is captured by the EWMA average), conditioned on which the more distant past matters very little. The extra amount of expressiveness and conservatism offered by maintaining an entire distribution is offset by the complication it instills into the reasoning and form of the second part of our controller, i.e. the forecast for the future.

The second part of our controller answers the following question: given that the current link rate (estimate) is λ, how many data can be trasmitted in the next 100ms, as a function $f(\lambda)$ of λ? In fact, we will make the simplifying assumption that $f(\lambda) = c\lambda$, for some constant c that does not depend on λ (but might vary with time as a result of new inferences). The justification for this assumption is that we think of the link rate as varying according to an *Exponential Brownian Process*, i.e. of the form e^X where the exponent X is a Brownian Process with a certain fixed variance σ. This way, for example, the probability that the rate goes from 10Mbits/sec to 20Mbits/sec in the space of a tick would be equal to the probability for it to go from 1Mbit/sec to 2Mbits/sec in one tick. It seemingly makes more sense than just *Additive Brownian Process*, for which the probability of going from 10Mbits/sec to 20Mbits/sec would equal to that of going from 1Mbit/sec to 11Mbits/sec. Furthermore, it also eliminates the nuisance of dealing with the rate going below zero. Obviously, our model for the link rate implies a homogeneous relation between the current link rate and the link rate at any percentile, at any fixed number of ticks later, which implies a homogeneous relation between the current link rate and the total throughput at any percentile, within any fixed interval of time immediately following, which can be captured by a single constant.

Rather than fixing a single value for c throughout the entire run, we update c periodically based on the empirical ratios we have seen during the run. At the end of each tick in the history of the run, there is one such ratio between the total throughput in the 100ms following that tick, and the link rate at that instant (which we approximate using the running average λ at that point). This ratio can indeed be calculated, as long as the corresponding tick is at least 100ms in the past, because then we would already have the measurement for the total throughput (in the 100ms interval after that tick), by now. To infer c, we look at the 50 most recent such ratios, and set c conservatively equal to their 90th percentile value. In other words, we set c so that, based on empirical observation, we are 90 percent certain that the predicted total throughput for the next 100ms will not exceed what is actually achieved. We re-estimate c roughly once every second ($50 \times ticklength$).

There is one more point to clarify. We make our congestion window size to be capped below at 10. The reason is we observe that when the window size gets too small during the run, it is likely to just get stuck there even after the channel condition improves. This is probably because the sender is receiving very few acks (because it is sending out very few packets), and so the observed ratios described in the previous paragraph remain small, leading to a small updated window size... i.e. a vicious cycle. It seems that we just need to escape from the "fixed point" inherent to this cycle of feedback/update. Through experimentations, we observe that capping below at 10 enables this escape and allows the estimates to be pulled alongside the actual channel condition.

In the end, with this scheme we achieved an average throughput of 3.67 mbps and average signal delay of 152, for a score of 24.14. Figures 2 and 3 show plots of throughput and signal delay over time.

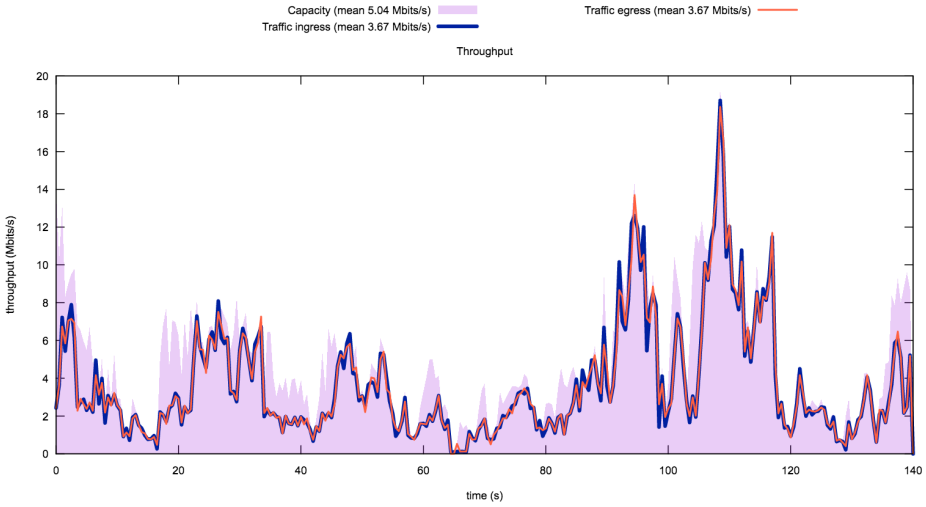

Figure 2: Throughput over time

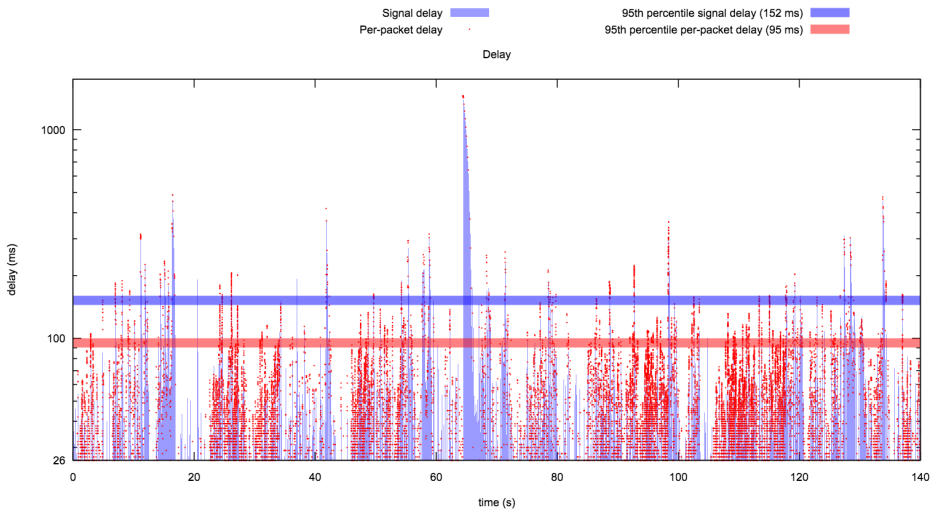

Figure 3: Throughput over time