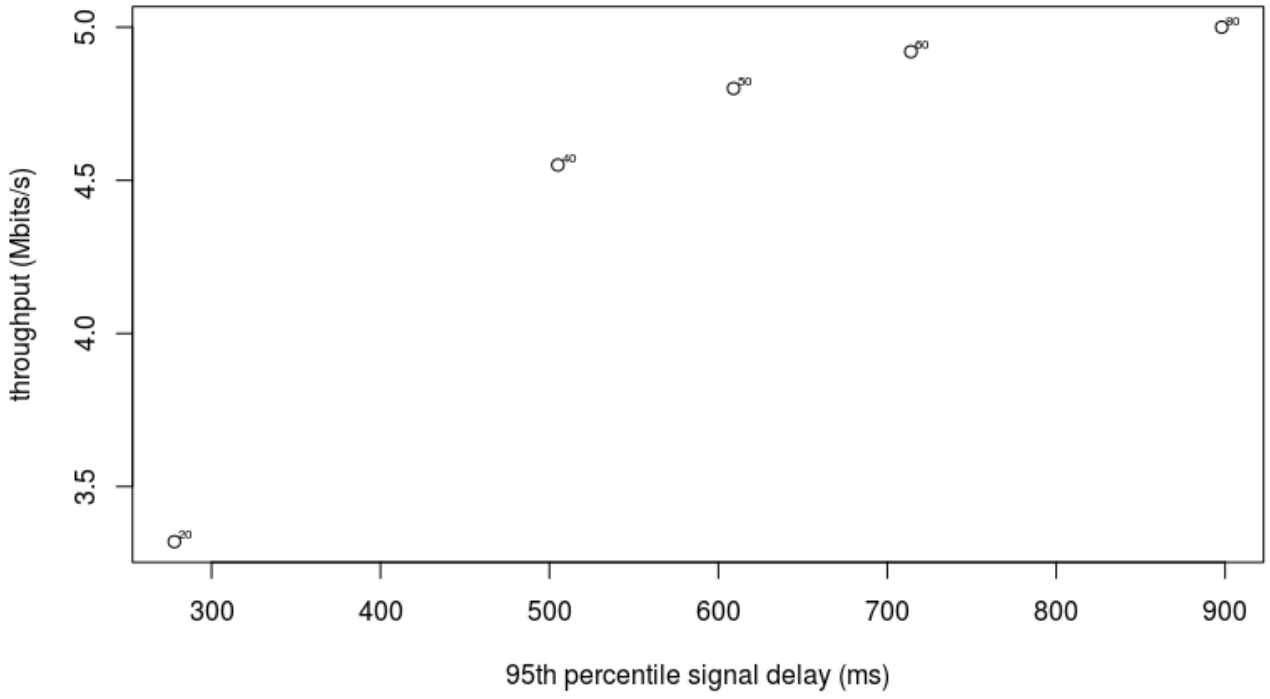
Merry Mou
10/24/16

6.829 Pset 2 Report

Warmup exercise A

3 measurements per window size (20, 40, 50, 60, 80) were taken. The measurements had very small variation (aka, the measurements were very repeatable), so an average is plotted in the graph below. The best window size (which maximizes log(throughput/delay)) is 20.



Warmup exercise B

See branch “ex-b” for work.

After a bit of tweaking, I chose an initial window size of 10. The signal to increase the window size is th receipt of a packet of greater sequence number than we have seen before. The signal to decrease window size is seeing 1 duplicate sequence number ack.

This still did not produce great results. Average throughput is 5.03 Mbits/s (99.8% utilization), with a 95th percentile per-packet queueing delay of 56571 ms.

Warmup exercise C

See branch “ex-c” for work. We use the rtt for each packet received to update the window size. If rtt is above some max delay, then decrease the window size by half. Otherwise, increase window size by 1. Experiment with a few thresholds or tweaks and report on what worked the best.

Max delay values in the range of 100ms to 900ms all caused average throughput to be around 60-70% utilization, and smaller and larger delay values both caused increases in signal delay. So something

around 500ms worked best (average throughput of 3.06 Mbits/s (60.7% utilization), and 95th percentile signal delay of 2187 ms).

Exercise D:

Understanding Mobile Networks

After playing around with a couple of congestion control algorithms, we notice a couple of things. Limits in mobile bandwidth are due to two main reasons: capacity limits and variability of network quality. A generalization would be that capacity limits are better reflected by packet drops (number of packets received per some time period), while network quality is is better reflected by round trip time (RTT).

Also, simple AIMD is probably not sufficient for mobile networks for several reasons. First of all, it is too sensitive to individual packets' behavior, and other noisy rises and falls. Hence, the appeal of EWMAs or something else to smooth out behavior. Also, the addition and multiplication parameters are not flexible enough to adapt to highly variable networks. e.g., if the network suddenly completely cuts out, it might take several RTTs until the window size decreases enough. We need to aggressively prevent high delays, and we need better ways of accounting for random variability of mobile networks.

With this motivation, we implement a congestion control algorithm that uses both number of packets received and RTT to increase or decrease the window size per tick.

Design
For every tick, we count the number of packets `num_packets` to arrive in each tick, and keep a moving average of the per packet RTT `rtt`.

Every tick, we update the window size `the_window_size`.

We increase the window size if `num_packets` is big (`>= the_window_size`) or `rtt` is small enough. The window size is increased by basically taking a weighted average of the `num_packets` and `the_window_size`.

We decrease the window size if `num_packets` is small (`<= the_window_size`) or `rtt` is big enough. The window size is decreased to basically a fraction of the weighted averagef the `num_packets` and `the_window_size`.

We kept the rule to recover from timeouts (no packets received in a tick): if we go by 1 ticks without seeing any packets, send 1 packet. Also, because of the negative feedback loop where a small window size causes us to see few packets, we have a “restart” rule, where if `num_packets` is small and nonzero (it will be nonzero because of the timeout recover rule), and `the_window_size` is small, and `rtt` is small enough, reset `the_window_size=30.`

All boundaries have a margin (e.g. “if `num_packets >= c*the_window_size`”, where c is some constant, say `1.1`), to allow for noise.

Results

See branch “ex-d” for work.

Average capacity: 5.04 Mbits/s
Average throughput: 4.79 Mbits/s (95.0% utilization)
95th percentile per-packet queueing delay: 122 ms
95th percentile signal delay: 191 ms

Discussion

Using RTT gives us more information about the state of the network. For example, if we are receiving a lot of packets, if those packets have a high RTT then we should not increase the congestion window. Using a moving average of RTT helps us adapt to changing network conditions, but self-inflicted delay that drives up RTT will not cause a positive feedback loop that drives up window size (and drives up self-inflicted delay and RTT), since window size is also dependent on number of packets received.

Also, updating regularly for each time interval allows us to not rely on the timing of arriving packets to update the window size, allows us to adapt quickly to changing network conditions, and using metrics per time interval to update the window size helps smooth out window size updates. So after some tweaking, we choose a tick interval that is not too small or too large (100 ms).