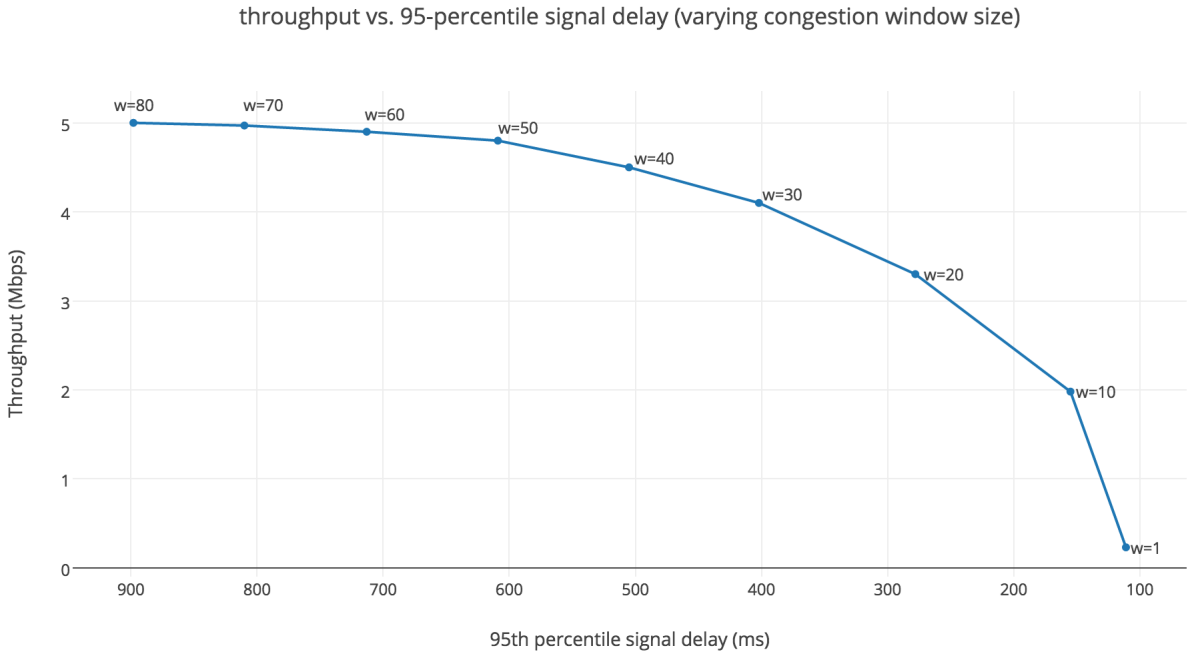
6.829 Problem Set 2

Eric Manzi (ermanzi@mit.edu)

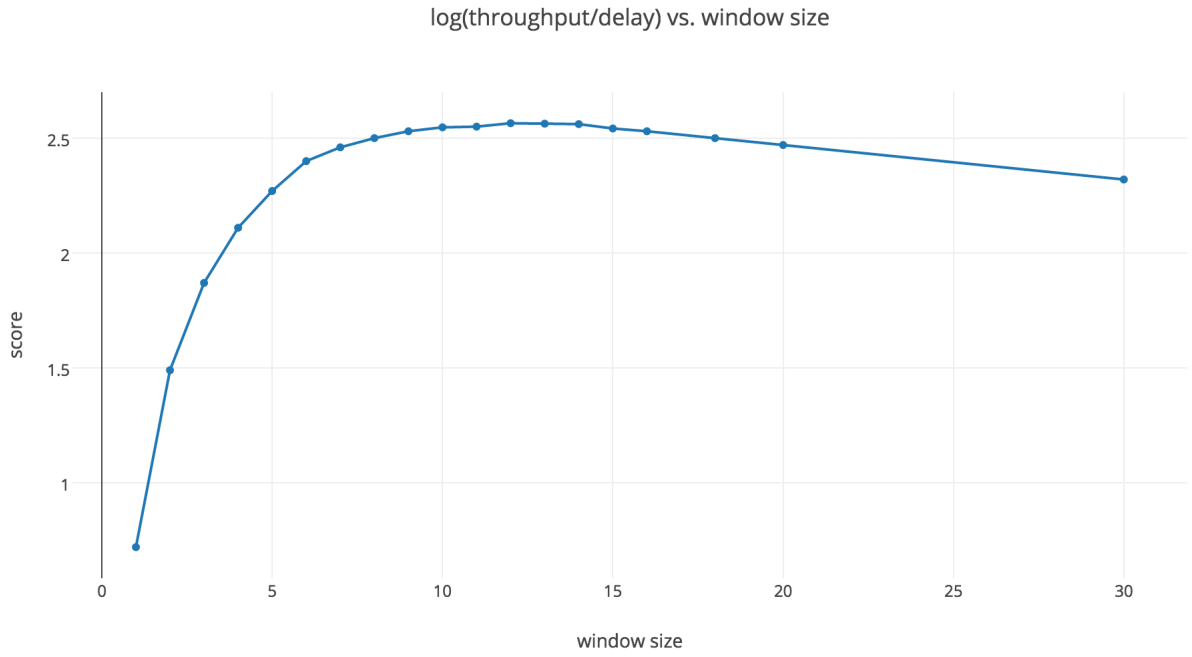
Oct 24, 2016

1 Exercise A: Varying fixed window size

The graph below shows throughput vs delay for fixed window sizes from 1 to 80 in increments of 10:



It seems the score $log(\frac{throughput}{delay})$ is maximized when w is in the range $5 \leq w \leq 20$. I repeated measurements for all values of w from 1 to 20 and found that w=12 yielded the maximum score that was observed, 2.565. I run these measurements two more times for $8 \leq w \leq 16$ and found that w=12 always yielded the maximum score although this score varied slightly. The results are shown in the plot below:



2 Exercise B: AIMD

Algorithm: My initial implementation increased the congestion window by 1 on each ACK and decreased it by a factor of 2 when the RTT exceeded the timeout 1000ms.
Evaluation: How well does this work? This scheme poorly especially with regards to delay. I made the following changes to improve it:

1. The timeout is too high, which allows congestion to build up before it is noticed. I set the timeout to roughly $2 \times RTT_{min} = 90$ after trying a few different values, with other constants fixed. My RTT_{min} was 42ms, so my timeout was 90ms for this trace.

2. When a multiplicative decrease happens, it takes some time before the effects of the new window size can be noticed, so I added a variable `num_acks_til_next_md` which is set to the current window size multiplied by some factor, md_interval_factor. This variable is decremented by 1 on each ACK until it is 0, at which point the window can be decreased again if RTT exceeds timeout.

3. Instead of incremented the window size by 1, increment it by $\frac{ai}{cwnd}$. I tested different values for the additive increase constant ai and multiplicative decrease factor md and found that $ai = 2$ and $md = 2$ gave me the best power score $(3.92/0.115) = 34.06$ (this varied between measurements).

3 Exercise C: Simple Delay-triggered scheme

Algorithm: The scheme performs a multiplicative decrease if the rtt is over the upper threshold, an additive increase (cwnd+=ai/cwnd) if it is between the lower and upper thresholds, and an aggressive increase (cwnd+=ai) if it is under the lower threshold.
Evaluation: I found that this scheme worked best for me when the lower threshold value set to $1.1 \times RTT_{min}$ and the upper threshold value set to $2.4 \times RTT_{min}$. Setting the additive increase constant to 1 and the multiplicative decrease factor to 2 gave me the best average throughput of 3.73 Mbps and a 95th percentile signal delay of 109 ms, a power score of $\frac{3.73}{0.109} = 34.2$ (this varied between measurements)

4 Exercise D: Contest

I started by making some improvements to the delay-triggered scheme.

- Since the link capacity is spiky, increasing the window size by a constant additive increase constant ai does not take advantage of the high capacity when it is available. I increase ai by a factor of 1.005 on each ACK when rtt $<$ timeout and reset it to 1 when rtt $>$ timeout, where $timeout = 2 * RTT_{min}$. This increased my power score by +0.5.

- Since waiting for ACKs and checking if it exceeded the timeout takes too long, another minor change I made was to look at the send times for all datagrams since the last ACKed and check if the difference between current time and send time exceeds timeout to signal congestion.
Now waiting for `1.5*window_size` ACKs before the next multiplicative decrease we wait for `last_sequence_number_se` ACKs. This increased my power score by +0.7 although this might have been variation between measurements.

- Another improvement I tried was to decrease the window size by an amount proportional to the $RTT_i/2 \cdot RTT_{min}$ so that if the RTT_i is much greater than the timeout, the window size drops substantially.

- I then use the exponential weighted moving average of the changes in RTT, rtt_{delta}. If this is negative, that means the queue is draining, so if the rtt is below the threshold, the window should increase more aggressively. If the rtt is over the threshold, then the window should decrease less aggressively if at all. Conversely, if $rtt_{delta} > 0$, the queue is filling up and we don't want to increase the window size if rtt is below the threshold but we want to decrease the window size even more aggressively if rtt is above the threshold. My final score was: 36.2. Running this scheme against the TMobile traces gave an average throughput of 8.15 Mbits/s and a 95th percentile signal delay of 120 ms.