6.829 Pset 2

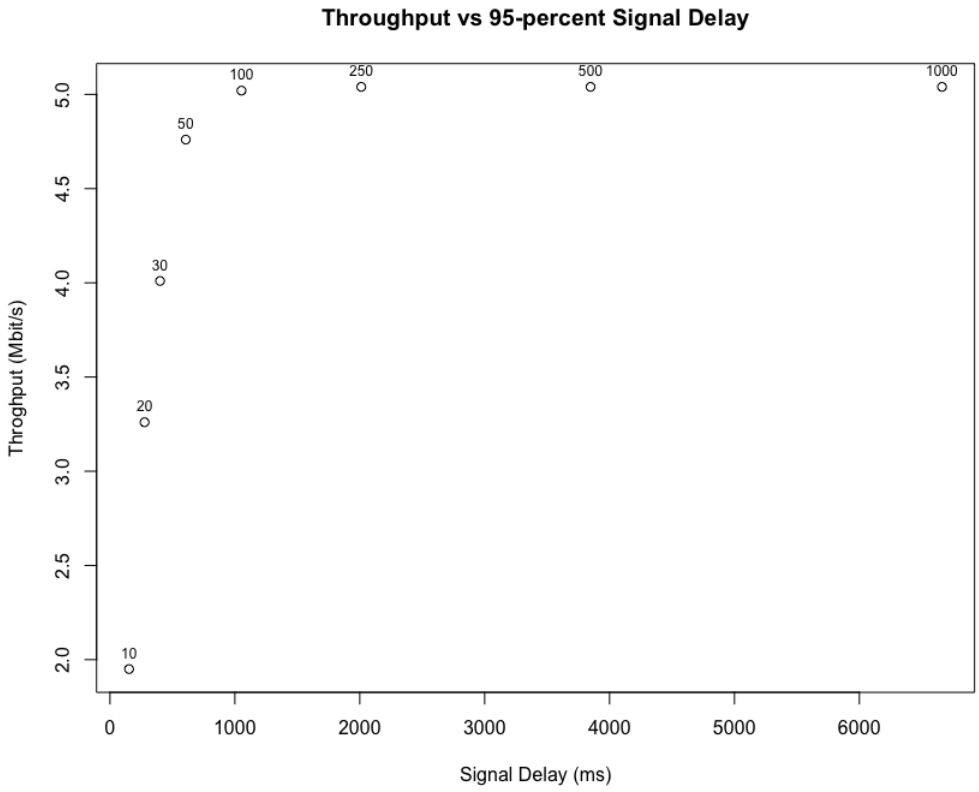
Carlos Caldera and Lilika Markatou

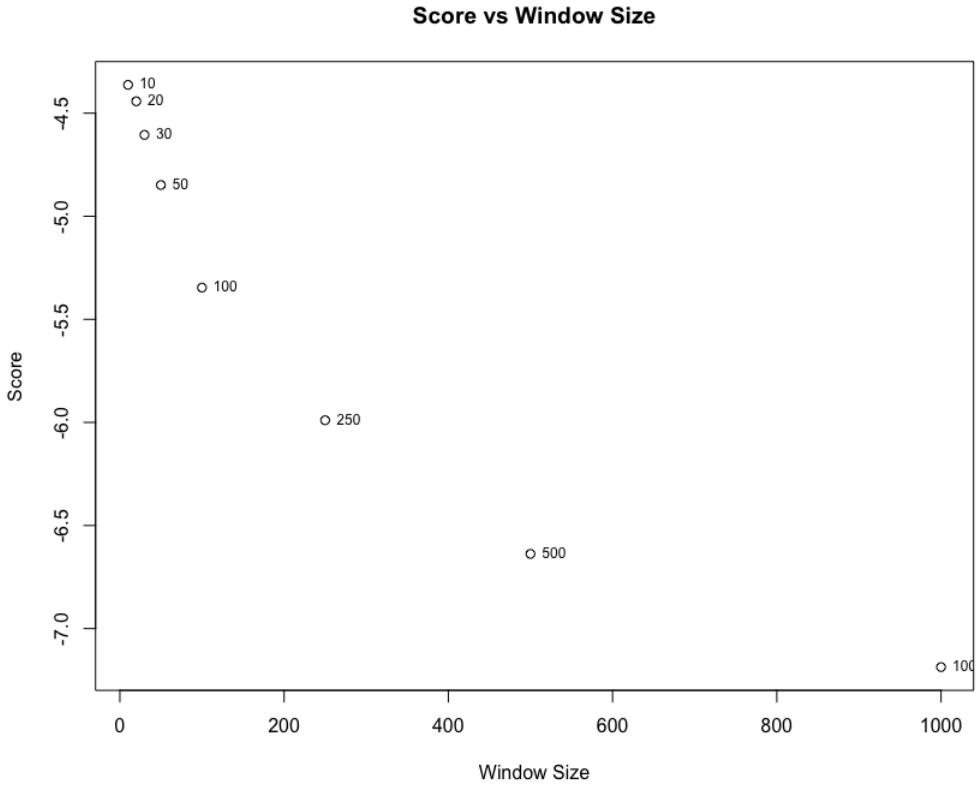
October 2016

Exercise A

The best window value we found was a window size of 12. Because we don't vary our window size in this example and the process is fairly deterministic, the measurements taken with the same window size are very repeatable.

The following graphs show our measured throughputs vs delay for our changing window size (the window sizes are indicated), as well as score vs window size, where score = $\log(\frac{throughput}{delay})$


Throughput vs 95-percent Signal Delay
Throghput (Mbit/s)
Signal Delay (ms)
2.0
2.5
3.0
3.5
4.0
4.5
5.0
0
1000
2000
3000
4000
5000
6000
10
20
30
50
100
250
500
1000



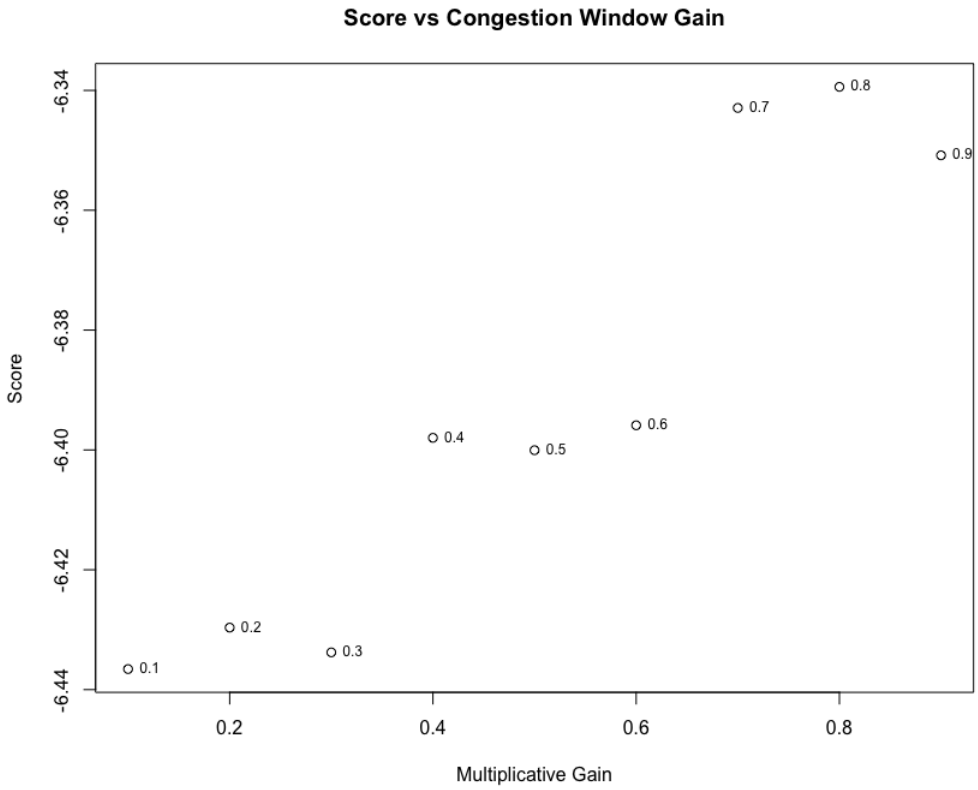
Exercise B

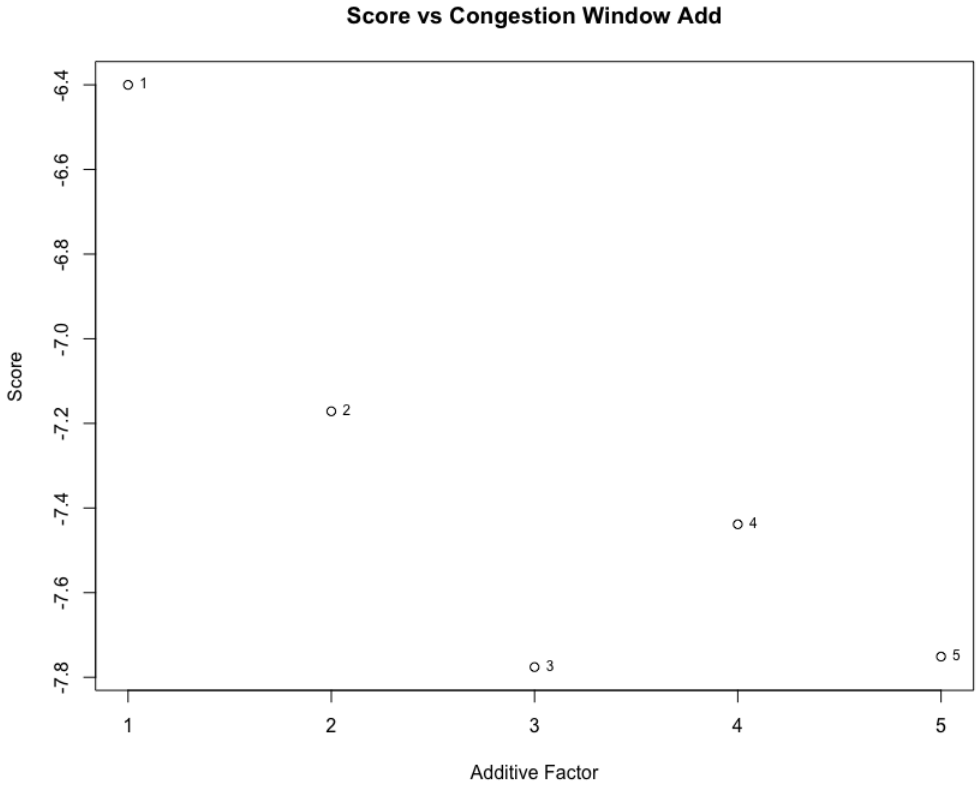
A simple AIMD scheme did not do as well as one would hope (it actually did worse than a simple constant-sized window). For our AIMD scheme, we tried varying two factors: (1) the amount of gain on congestion (i.e., the constant in MD), and (2) the amount of of additive increase on no congestion. We used the optimal value we found for (1) for (2). Coincidentally, our AI constant was 1, and our MD constant was 0.5.

Code for this protocol can be found at:

https://github.com/markatou/sourdough/tree/exercise_B

The following graphs show our results for (1) and (2), respectively. Again, the score = $\log(\frac{throughput}{delay})$


Score vs Congestion Window Gain
Score
-6.34
-6.36
-6.38
-6.40
-6.42
-6.44
0.2
0.4
0.6
0.8
Multiplicative Gain
0.1
0.2
0.3
0.4
0.5
0.6
0.7
0.8
0.9



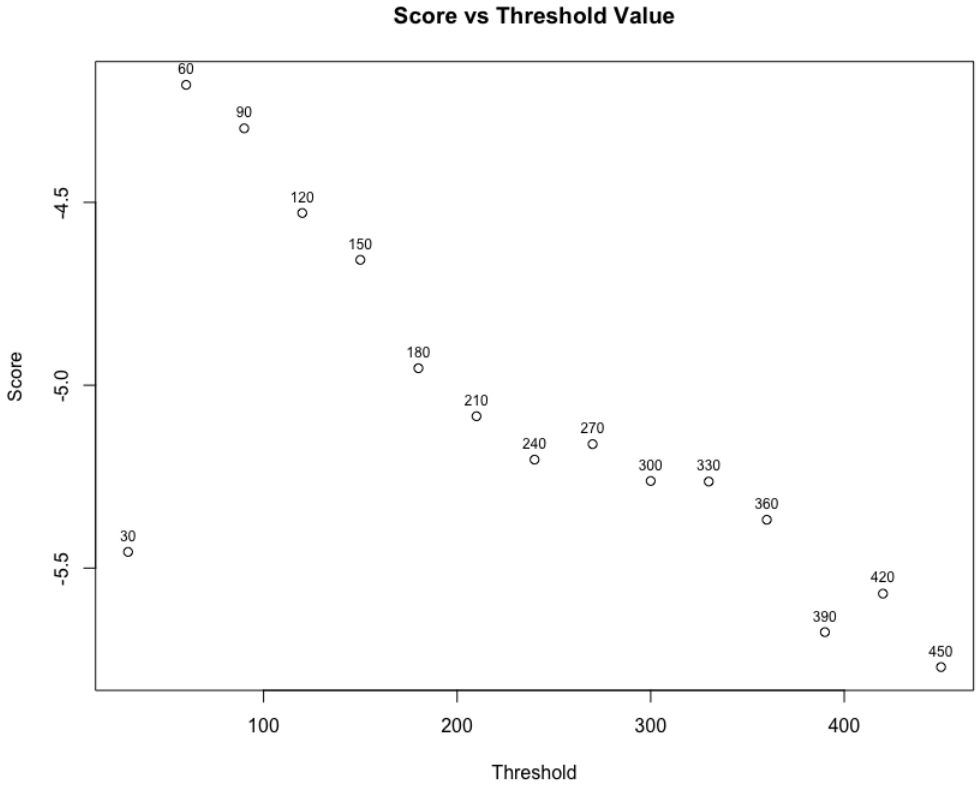
Exercise C

For our simple delay-triggered scheme, we tested threshold values in the range from 60ms to 450ms, at 30ms intervals. Lower threshold values gave better results (with the exception of 30ms); our best value was at a threshold value of 60ms.

Code for this protocol can be found at:

https://github.com/markatou/sourdough/tree/exercise_C

The following graphs show our results. Again, the score = $\log(\frac{throughput}{delay})$



Exercise D

In order to create our best approach we needed to work in two fronts. First, we had to find a good signal for congestion and then we had to develop the best way to deal with that signal.

We tried a number of different signals for determining how congested the network was:

1. A packet being dropped. (Exercise B)
2. Whenever the roundtrip time (RTT) of the last packet is bigger than a constant threshold. (Exercise C)
3. Our best results come from keeping a weighted average of the roundtrip time (RTT) of the packets. When the RTT of the last packet is bigger the weighted average of the previous RTTs plus a constant, the window size decreases, otherwise it increases. We also modify the timeout. Instead of having a constant, we use a weighted average of the RTTs.

We tried a number approaches to deal with the congestion signal.

(a) AIMD

This approach halves the window size in the face of congestion, and increases it by one all other times. It achieves the best results from the techniques we tried. When run on Verizon-LTE-short, it achieves a utilization of 85%, with a 95th percentile signal delay of 156 ms, getting a power score of 27.

(http://cs344g.keithw.org/report/?acai-1477620341-huoniedu)

Code for this technique is located on the master branch of

https://github.com/markatou/sourdough.git

(b) AIAD

We decided to try an additive increase, additive decrease approach. This approach achieves a utilization of 93% with a 95th percentile signal delay of 219 ms, achieving a power score of 21.37.

(http://cs344g.keithw.org/report/?strawberry-1477621918-eiwoobei)

This protocol needed to be overfitted in order to work properly and a small difference in the constants can be catastrophic in terms of power score.

Code for this can be found at

https://github.com/markatou/sourdough/tree/AIAD.

(c) MIMD

Finally, we tried a multiplicative increase and multiplicative decrease technique. This techique appears to be following the capacity of the channel fairly well, but it often fails unexpectedly. It achieves a 25 % utilization with a 95th percentile signal delay of 269 ms and a power score of 4.72. (http://cs344g.keithw.org/report/?strawberry-1477622765-yuaphaek)

Code for this protocol can be found in

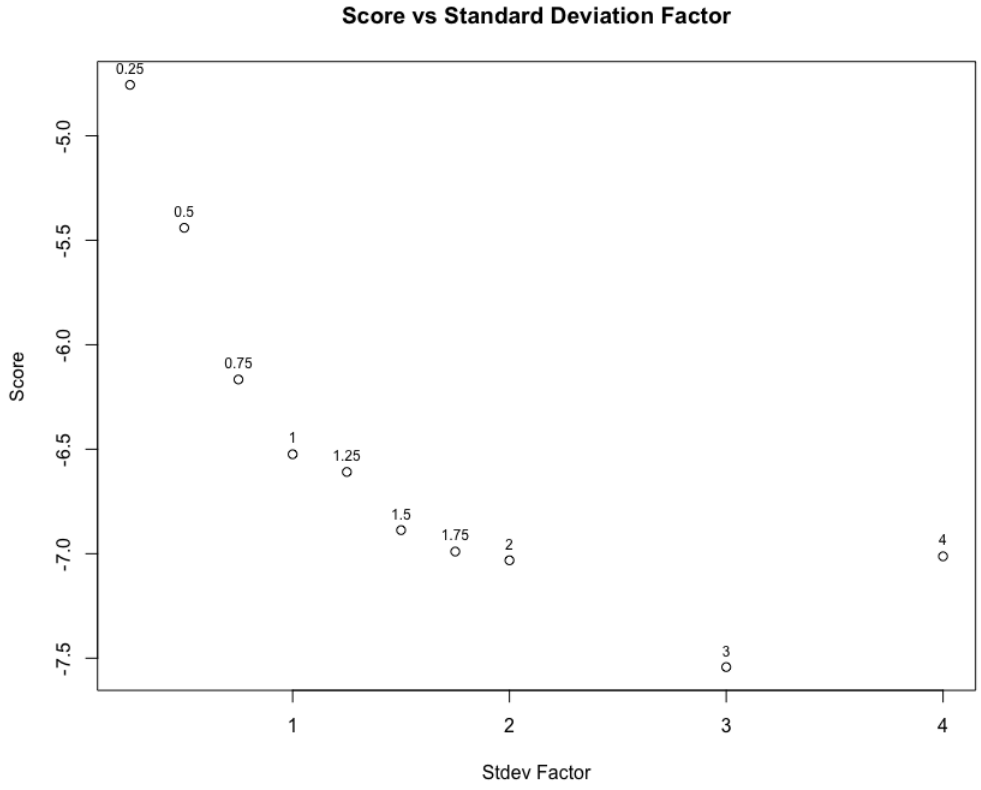
https://github.com/markatou/sourdough/tree/MIMD

4. We assumed a normal distribution of the RTTs, and choose our congestion signal to be whenever the RTT of the last packet was greater than the mean plus a standard deviation (times a constant we changed).

   Code for this protocol can be found at:

   https://github.com/markatou/sourdough/tree/exercise_D_4

   The following graphs show our results, where we varied the constant factor for standard deviation. Again, the score = $\log(\frac{throughput}{delay})$

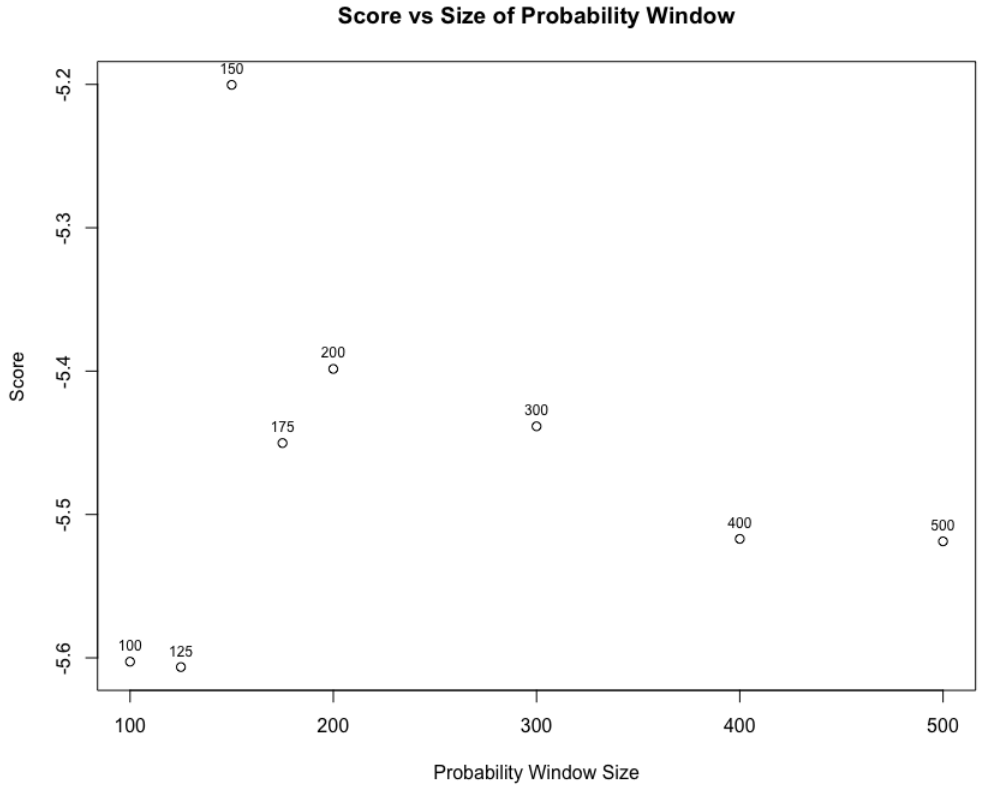


5. We used the change in RTT to estimate the change in capacity (i.e., the larger the delay gets the less capacity we probably have) and used that information to probabilistically change our window size. We did this by having a probability window of some fixed size, with a threshold value that would change depending on how we thought the capacity was changing. Then, a random integer was drawn in the range $[1, size(probability\_window)]$ and compared that with the current threshold to determine whether we increased or decreased the window size.

   Code for this protocol can be found at:

   https://github.com/markatou/sourdough/tree/exercise_D_5

   The following graphs show our results, where we varied the size for our probability window. Again, the score = $\log(\frac{throughput}{delay})$



In order to make sure that our protocol, located in the Master branch of the repository, was not overfitted, we run it on the TMobile-LTE-short and achieved an average throughput of 14.83 Mbits/s (88.6% utilization) with a 95th percentile signal delay of 259 ms.

In order to run the code please run ./run.sh in sourdough. Our different attempts are located in different branches of the repository, with our preffered one in the main branch.