CS 244 Assignment 2

Brad Girardeau
bgirarde@stanford.edu

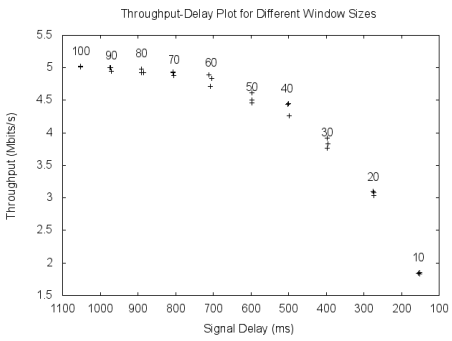
Samantha Steele
sammyst@stanford.edu

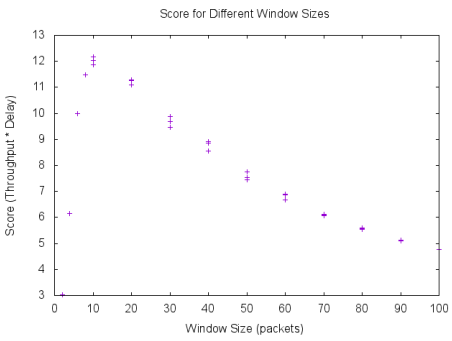
April 27, 2017

Github Repository: `https://github.com/sammyas/special-couscous`

All plot generation and data gathering scripts are in the `datagrump/tuning` subdirectory.

Exercise A



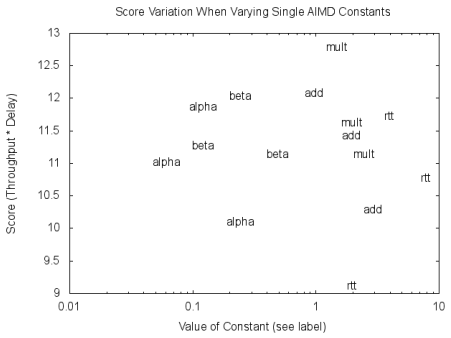


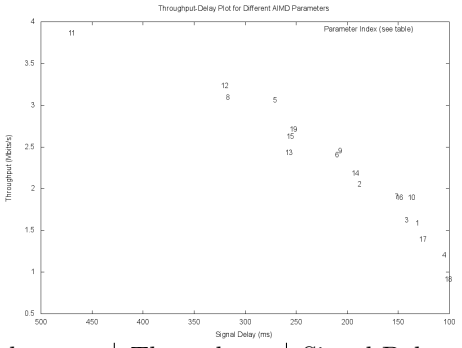
The best single fixed window size to maximize the score was 10 packets. As can be seen in the above graphs, the measurements with the same window size were fairly repeatable over multiple runs, clearly clustering together.

Exercise B

Overall AIMD does not perform well, only slightly outperforming a small fixed window. We tested a variety of different AIMD constants, ultimately finding the TCP alpha (0.125) and beta (0.25) values worked best for RTT estimation with an additive increase of 0.5 (TCP uses 1) and multiplicative decrease of 1.5 (TCP uses 2) with RTT timeout variance gain of 6 (TCP uses 4), giving a score of 13.89.

Details on test results can be seen in the following figures. First each parameter value was varied individually as seen in the left figure. The default TCP alpha and beta values were kept since they appeared near their peak, then the other parameters were varied together in the second figure. Index 10 had the highest score, so its values were used.

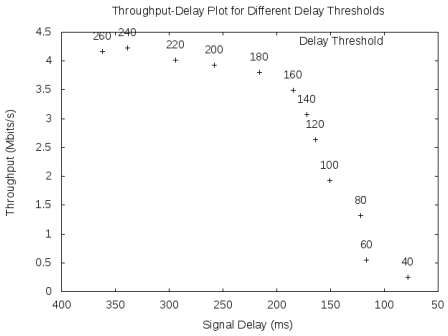


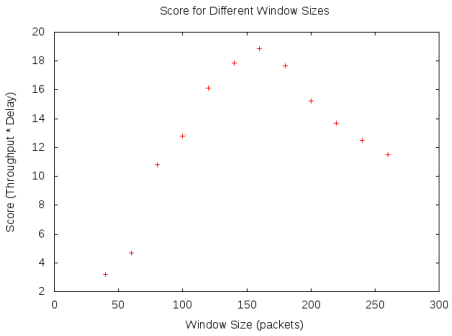


Index	RTT gain	Add gain	Mult decrease	Throughput	Signal Delay
1	4.0	1	2	1.58	131
2	4.0	0.5	1.125	2.05	187
3	4.0	0.5	1.25	1.62	141
4	4.0	0.5	1.5	1.2	104
5	4.0	1	1.125	3.06	270
6	4.0	1	1.25	2.4	209
7	4.0	1	1.5	1.9	151
8	6.0	0.5	1.125	3.09	316
9	6.0	0.5	1.25	2.45	206
10	6.0	0.5	1.5	1.89	136
11	6.0	1	1.125	3.86	469
12	6.0	1	1.25	3.23	319
13	6.0	1	1.5	2.43	256
14	4.0	2	2	2.18	191
15	4.0	3	2	2.62	255
16	4.0	1	1.5	1.89	148
17	4.0	1	2.5	1.39	125
18	2.0	1	2	0.91	100
19	8.0	1	2	2.71	252

Exercise C

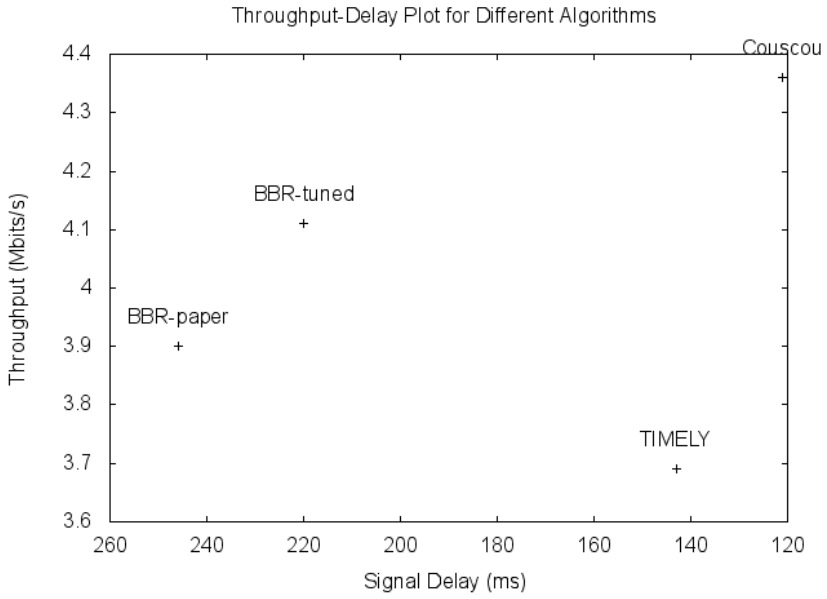
We combined a delay threshold with the AIMD model, decreasing the window multiplicatively when the RTT passed the threshold and otherwise increasing the window additively. The default TCP additive gain of 1 and multiplicative decrease of 2 were used. We tried a variety of fixed window thresholds as seen in the figures below, and the best value was 160, giving a score of 18.86.





Exercise D

We experimented with three main approaches to effective congestion control. Our first algorithm was modeled on the Google BBR paper [1]. However, BBR did not adapt dynamically enough to rapid bandwidth changes. Our second approach mirrored the RTT based approach of the TIMELY paper[2]. Finally, we experimented with a hybrid approach. We used a very simplified version of BBR that estimates bandwidth via exponentially weighted moving averages(EWMAs) as in TIMELY instead of the min/max filters used by traditional BBR. This resulted in our highest power score of 36.60 (3.55 Mbits/s throughput and 97ms signal delay). Code for our three approaches is stored on github branches bbr, timely and master. The following figure summarizes the throughput/delay performance of each algorithm.



BBR relies on both rate based sending and a window based inflight limit. As a result, we modified sender.cc to send packets at a specified rate. We also implemented filters to track the running min/max of $BtlBw$ and RT_{prop}. At first, we initialized BBR with a fixed $BtlBw$ rather than implementing the exact BBR startup behavior. This led to very high delay (300-400ms). We hypothesized that this was caused by large standing queues. To solve this, we added STARTUP and DRAIN phases. These reduced signal latency to 246 ms. We did parameter tuning to optimize the window size of our filters. A BtlBw window of 100 was optimal. For RT_{prop}, we found that using a single min rtt counter gave lower delay than a windowed filter. This lowered signal delay to 215ms.

Despite the decrease in latency, our throughput remained low. In examin-

ing graphs, we realized our algorithm was very slow to respond to increases in bandwidth. To improve this, we experimented with the configuration of pacing stages. The BBR paper uses pacing stages: [5/4, 3/4, 1, 1, 1, 1, 1, 1]. To get faster window growth, we found the best throughput with [5/4, 3/4, 1, 1, 5/4, 3/4, 1, 1]. After all these modifications, our power score was still only 18.6.

Next, we pivoted to an rtt based approach following the TIMELY algorithm. TIMELY calculates the RTT using an exponentially weighted moving average and calculates a delay gradient based on Δrtt and rtt_{min} to capture changes in RTT. If the RTT is below a min threshold or above a max threshold, the algorithm follows AIMD. Otherwise, the window is modified based on the current gradient. This algorithm required optimizing four parameters: T_{low}, T_{high}, α and β. We used a grid search to optimize T_{low}, T_{high} and then did a second search to optimize α and β. Our optimal parameters were: $T_{low} = 80$, $T_{high} = 200$, $\alpha = 0.125$, and $\beta = 0.8$. This gave our highest power score yet of 28.8.

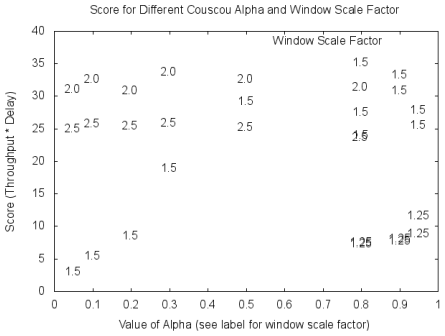
TIMELY's use of an EWMAs to calculate RTT was successful because it adapted well to oscillating bandwidth. We wondered if this could be applied to estimate bottleneck-bandwidth for BBR. This decision was also motivated by the use of EWMAs in the Sprout paper. Rather than use filters to estimate BtlBw, we used an exponentially weighted moving average. This removed the need for filters– as described above, we were already using a single min rtt calculation for RT_{prop}. Since rate based sending and pacing stages phases added complexity without tangible performance gains, we decided to eliminate these. This resulted in a much simpler algorithm that estimated window size:

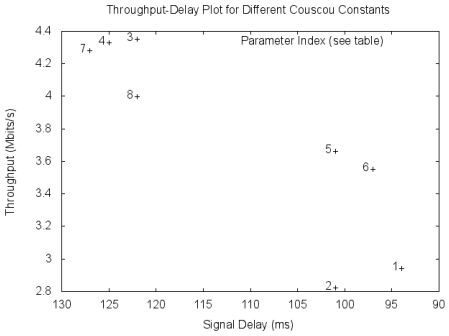
$$cwnd = w * btlbw_{avg} * rtt_{min} \tag{1}$$

Here, w is the window scaling constant. This is one of three parameters in this model. The other two are α, which controls the exponential moving average and the timeout (in ms) after which a packet will always be sent.

We first did a grid search over w and α to identify promising values as seen in the left figure below (timeout was 100ms by default). Higher α (which corresponds to weighting the current bandwidth estimate more) improved performance when paired with a lower window size scaling factor. This made sense because the model could then react faster to bandwidth changes and maintain a more accurate sending window size.

With a narrowed range of w and α, we then searched for the combined optimum setting with the timeout parameter included. The results of these trials are in the figure on the right below, with corresponding table. The best parameters identified ($\alpha = 0.9$, $w = 1.5$, $t = 75$) gave an overall best score of 36.60.





Index	Alpha	Window Scale	Timeout	Throughput	Signal Delay
1	0.8	1.5	50	2.94	94
2	0.8	1.5	75	2.82	101
3	0.8	1.75	50	4.35	122
4	0.8	1.75	75	4.33	125
5	0.9	1.5	50	3.66	101
6	0.9	1.5	75	3.55	97
7	0.9	1.75	50	4.28	127
8	0.9	1.75	75	4.0	122

Exercise E

COUSCOU: Congestion Obviation Under Substandard Cellular Operational Utilities

References

[1] Cardwell, Neal, et al. *BBR: Congestion-Based Congestion Control.* Queue 14.5 (2016): 50.

[2] Mittal, Radhika, et al. *TIMELY: RTT-based Congestion Control for the Datacenter.*. ACM SIGCOMM Computer Communication Review. Vol. 45. No. 4. ACM, 2015.