CS 244 **Spring 16-17**

Assignment 2: Congestion Control Contest

Professors: K. Winstein & S. Katti *Students: Jervis Muindi & Luke Hsiao*

Exercise A

Question 2.1 *Vary the fixed window size by editing controller.cc to see what happens. Make a 2D graph of throughput vs. 95-percentile signal delay (similar to what is seen on the contest analysis URLs) as you vary this value. What is the best single window size that you can find to maximize the overall "score" (log throughput/delay)? How repeatable are the measurements taken with the same window size over multiple runs?*

We tried several different windows sizes in increments of 5 packets. The raw numbers are shown in Table 2.1. Score is calculated as *throughput/delay*, and the best score is shown in bold.

Table 2.1: Throughput and Delay vs Fixed Window Size

Window Size	Throughput (Mbits/s)	95% Signal Delay (ms)	Score
5	1.05	109	9.63
10	1.93	155	12.45
15	**2.66**	**212**	**12.55**
20	3.26	277	11.77
25	3.73	343	10.87
30	4.07	401	10.15
35	4.32	453	9.54
40	4.51	504	8.95
45	4.65	557	8.35
50	4.76	607	7.84
55	4.85	652	7.44
60	4.91	711	6.91
65	4.94	763	6.48
70	4.96	808	6.13
75	4.98	855	5.82
80	4.99	896	5.57
85	5.00	935	5.34
90	5.00	972	5.14

These values are plotted as a 2D graph of throughput vs 95-percentile signal delay in Figure 2.1.

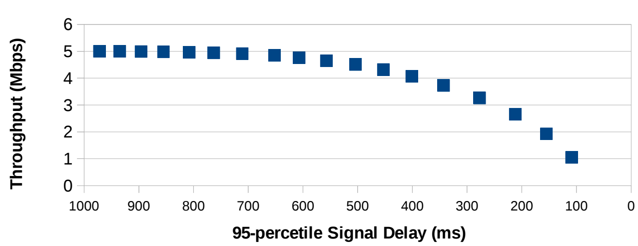

Figure 2.1: Throughput vs 95-percentile Signal Delay for varying fixed window size.

To answer the questions, we found that the best score was at a fixed window size of about 15 packets. The measurements taken with the same window size over multiple runs were very repeatable, in the mahimahi environment. The throughput and delays only varied slightly (i.e. ± a few hundredths of Mbps or a few ms).

Exercise B

Question 2.2 *Implement a simple AIMD scheme, similar to TCP's congestion-avoidance phase. How well does this work? What constants did you choose?*

For this exercise, we implemented a simple AIMD protocol, which increments the `cwnd` by $alpha/cwnd$ on each ack, and decreases the window by `beta` on a loss event (i.e. $cwnd = cwnd/beta$). This mimics the AIMD behavior of TCP in congestion avoidance. In our mahimahi environment, there is no loss, and there is unbounded queues. Thus, to signal when a multiplicative decrease should occur, we use a fixed timeout value as a signal. We set the timeout to 80ms, which is approximately the 95-percentile signal delay on the top algorithms from the leaderboard. We also set the initial congestion window size to 10.

We tried a few of different values for `alpha` and `beta`, and recorded their performance in Table 2.2. We found that the typical values of `alpha` $= 1$ and `beta` $= 2$ perform worse than many of the fixed window sizes we tested in Exercise A. In particular, AIMD increased latency as the algorithm tries to slowly use all the throughput available. We then tweaked the constants to see how being more aggressive or timid is growth and decrease affected the power score.

Table 2.2: Throughput and Delay for various AIMD constants

Alpha	Beta	Throughput (Mbits/s)	95% Signal Delay (ms)	Score
1	2	4.72	847	5.57
1	3	4.68	765	6.12
1	4	4.65	734	6.34
1	5	4.64	724	6.41
1	10	4.59	705	6.51
0.1	1.1	3.80	377	10.08
0.1	**1.5**	**2.72**	**142**	**19.16**
0.1	2	2.43	186	13.06
0.5	2	4.41	570	7.74
0.2	4	3.23	267	12.10
2	2	4.88	1190	4.10
4	2	4.96	1655	3.00
4	10	4.95	1533	3.23

In general, we found that a slow additive increase (with `alpha` < 1), combined with a passive multiplicative decrease (`beta` < 2) provided a better power score on this cellular link that other configurations we tested. However, even in this best case, only about half of the available throughput was utilized. Even with the best constant values we found (shown in bold in the table) are not optimal.

Exercise C

Question 2.3 *Implement a simple delay-triggered scheme, where the window rises or falls based on whether the round-trip-time crosses some threshold value. Experiment with a few thresholds or tweaks and report on what worked the best.*

We implemented a simple scheme that increases or decreases the window size based on an RTT threshold. On each ACK, we calculate the RTT of the particular packet and additively increase the window if the RTT is less than the threshold, or additively decrease the window size if the measured RTT is greater than the threshold. The amount of increase and decrease are defined by the constant `alpha` and `beta`, respectively.

We experiment with several values of our three constants: `alpha`, `beta`, and `rtt_thresh`, and summarize the results in Table 2.3

Table 2.3: Performance of Delay-based Congestion Control

alpha	**beta**	**rtt_thresh (ms)**	**Throughput (Mbits/s)**	**95% Signal Delay (ms)**	**Score**
1	20	100	3.30	149	22.15
1	10	100	3.76	155	24.26
1	5	100	4.32	190	22.26
1	1	100	4.81	497	9.68
1	10	150	4.43	820	5.40
1	**10**	**90**	**3.57**	**137**	**26.06**
1	10	85	3.34	136	24.56
1	10	80	3.15	129	24.42
1	10	70	2.57	119	21.60
10	10	90	4.75	237	20.04
10	1	90	4.89	924	5.29
10	1	60	4.89	924	5.29

We found that using this approach, we achieved the best performance when the additive decrease occurred at a faster rate than the additive increase in order to quickly allow the queues to drain when a particular RTT is exceeded to reduce the queueing delay. We also found that the best RTT threshold was about 90ms.

Exercise D

Question 2.4 *Try different approaches and work to maximize your score on the final evaluation. Be wary about "overtraining": after the contest is over, we will collect new network traces and then run everybody's entries over the newly-collected evaluation trace. In your report, please explain your approach, including the important decisions you had to make and how you made them. Include illustrative plots.*

Exercise E

Question 2.5 *Pick a cool name for your scheme!*

Rocinante