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.

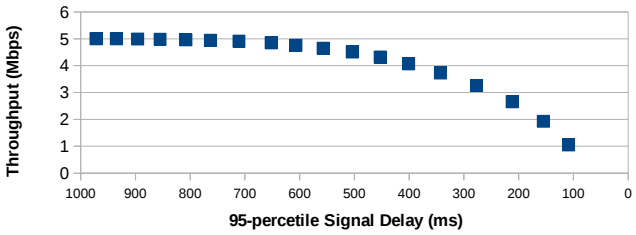

Figure 2.1: Throughput vs 95-percentile Signal Delay for varying fixed window size.

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. $\pm$ 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 se the timeout to 80ms, which is approximately the 95-percentile signal delay on the top algorithms from the leaderboard.

We tried a few of different values for `alpha` and beta, and recorded their performance in Table 2.2.

Table 2.2: Throughput and Delay for for various AIMD constants

Alpha	Beta	Throughput (Mbits/s)	95% Signal Delay (ms)	Score
1	2	XXX	XXX	XXX

We found that [stuff]

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.*

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!*