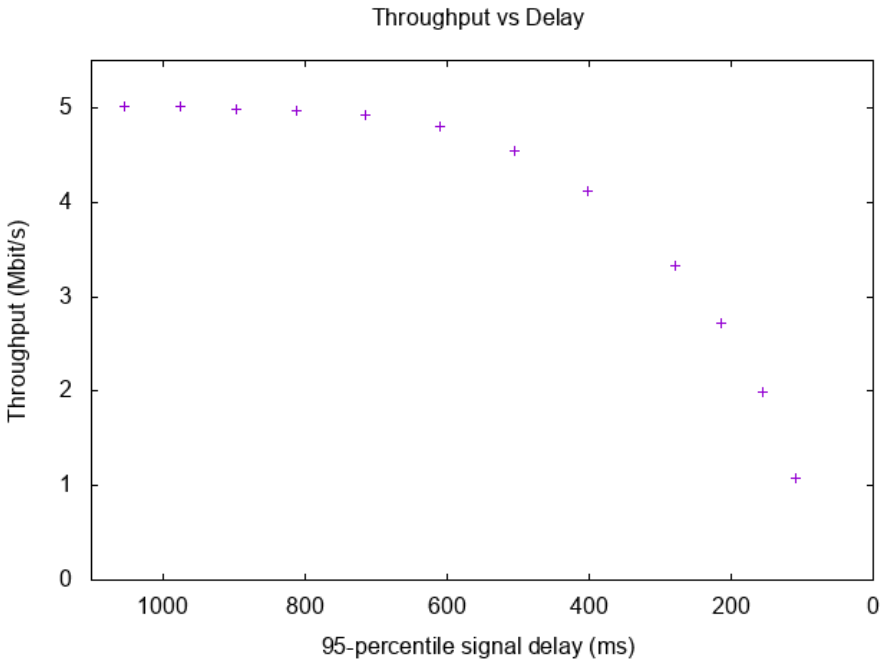
6.829 Problem Set 2

Anish Athalye / Kate Yu

Exercise A

Here is the data we collected

Window	Throughput (Mbit/s)	95-percentile delay (ms)	Score
5	1.08	108	10.00
10	1.99	155	12.83
15	2.72	214	12.71
20	3.32	278	11.94
30	4.11	402	10.22
40	4.55	505	9.01
50	4.80	609	7.88
60	4.92	714	6.89
70	4.97	811	6.13
80	4.99	897	5.56
90	5.01	975	5.14
100	5.02	1054	4.76



Best window size

We found that the best window size (among our data points) is 10 (for a score of 12.83).

Reproducability

We found the results to be very reproducible. We ran the simulation in a virtual machine on GCP (while not running anything else on the machine). When re-running the simulation with a window size of 50, for example, we got a throughput of exactly 4.80 Mbit/s and a 95-percentile delay of 609ms between runs.

Exercise B

We implemented a simple TCP-like AIMD scheme. Initially, we chose 1 as our additive increase parameter (A) and 2 as our multiplicative decrease parameter (B). That is, we increased our window size by 1/(window size) upon receiving

each ack and we decreased our window size by a factor of 2 every time we detected a dropped packet.

We did some experiments with varying A, B, and a hard-coded timeout parameter.

Timeout	A	B	Throughput (Mbit/s)	95-percentile delay (ms)	Score
1000	1	2	4.83	892	5.41
1000	3	2	4.95	1167	4.21
1000	1	4	4.84	893	5.42
500	1	4	4.73	480	9.85

Next, we implemented EWMA RTT estimation with weight EWMA (a parameter) and updated the implementation to consider a packet to be lost if no ack was received for LATE (another parameter) times the estimated RTT.

LATE	EWMA	A	B	Throughput (Mbit/s)	95-percentile delay (ms)	Score
1.5	0.1	1	2	3.42	612	5.59
1.5	0.2	1	2	4.69	1306	3.59
1.5	0.5	1	2	4.91	1648	2.98
1.1	0.2	1	2	0.92	115	8.00
1.2	0.2	1	2	1.45	123	11.79

In our experiments, AIMD didn't give an improvement over static window sizes. Compared to choosing a static window size (where we did a search over the window size space, perhaps overfitting the data), AIMD did worse at maximizing the target metric. However, AIMD can do pretty well at maximizing throughput (as we would expect).

Exercise C

We implemented a delay triggered scheme with a target delay centered around 100ms, with a 20ms tolerance in either direction. In terms of window size adjustment, we experimented with different schemes and parameters, as shown in the following table:

A corresponds to the additive factor, and B corresponds to the multiplicative factor.

Scheme	A	B	Throughput (Mbit/s)	95-percentile delay (ms)	Score
AIMD	1	2	3.04	134	22.69
AIAD	1		4.77	466	10.24
MIAD	1	2	4.94	18041	0.27

Scheme	A	B	Throughput (Mbit/s)	95-percentile delay (ms)	Score
MIMD		2	4.71	7873	0.59

Our AIMD-esque implementation performed well, resulting in a score of 22.69. We decided to further experiment with the AIMD scheme by tuning the parameters.

Lower (ms)	Upper (ms)	A	B	Throughput (Mbit/s)	95-percentile delay (ms)	Score
80	120	1	2	3.04	134	22.69
60	100	1	2	2.66	128	20.78
100	140	1	2	3.46	158	21.90
90	110	1	2	2.92	134	21.79
70	110	1	2	2.91	134	21.72
70	130	1	2	3.17	145	21.86
80	120	1	3	3.00	138	21.74
80	120	2	2	3.53	157	22.48

The best result was still achieved by the AIMD implementation centered around 100ms with a 20ms tolerance, a additive factor of 1, and a multiplicative factor of 2.

Exercise D

PID Controller

We implemented a PID controller to control window size to keep EWMAed RTT close to a given target. This isn't optimizing for exactly the right thing (we wanted to have a low 95th percentile signal delay), but it's still reasonably good.

We also added some code to prevent "reset windup", which is a condition that results in a large accumulated negative error. This is an issue with the system because window size can't be "driven" to a negative value.

We didn't really know a particularly principled way of setting PID parameters, so we manually tuned parameters. To avoid "overfitting" the PID parameters, we trained on the TMobile dataset and tested on the Verizon dataset. Here is the data from testing **on the TMobile dataset**:

Target (ms)	EWMA	K_P	K_I	K_D	Throughput	Delay	Score
90	0.2	1e-1	1e-2	2e-3	14.27	268	53.25
90	0.4	1e-1	1e-2	2e-3	14.29	268	53.32
90	0.2	1e-1	2e-2	2e-3	12.66	302	41.92

Grid Search

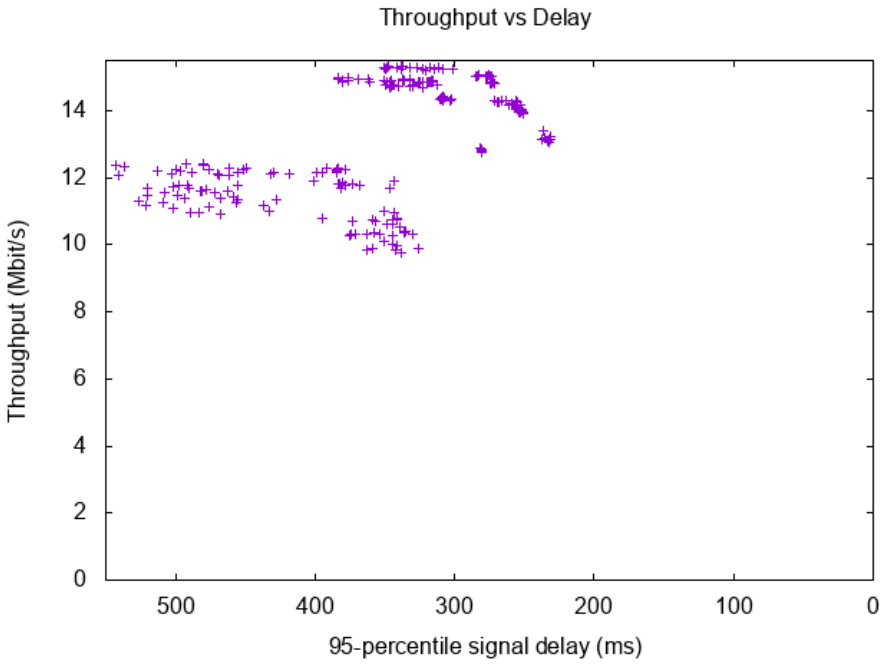
After this, we decided to do a grid search over a larger parameter space (see `gridsearch.py`):

```
SPACE = {
    'RTT_EWMA_FACTOR': ['0.2', '0.4'],
    'TARGET_DELAY': ['90.0'],
    'K_P': ['1e0', '5e-1', '1e-1', '5e-2', '1e-2'],
    'K_I': ['1e-1', '5e-2', '1e-2', '5e-3', '1e-3'],
    'K_D': ['5e-2', '1e-2', '5e-3', '1e-3', '5e-4'],
}
```

From 250 experiments, here are the bottom 5 and top 5 results:

EWMA	Target	K_D	K_P	K_I	Throughput	Delay	Score
0.4	90.0	1e-2	1e-2	1e-1	11.17	521	21.43
0.2	90.0	1e-2	1e-2	1e-1	11.31	526	21.50
0.4	90.0	1e-2	5e-2	1e-1	11.46	520	22.03
0.2	90.0	1e-3	1e-2	1e-1	11.11	502	22.13
0.2	90.0	1e-3	5e-2	1e-1	11.27	509	22.14
0.2	90.0	5e-4	1e-2	1e-2	13.14	232	56.63
0.2	90.0	5e-2	1e-2	1e-2	13.39	236	56.73
0.4	90.0	5e-2	1e-2	1e-2	13.40	236	56.77
0.2	90.0	5e-3	1e-2	1e-2	13.16	231	56.96
0.2	90.0	1e-2	1e-2	1e-2	13.24	231	57.31

Here's a scatter plot of throughput versus delay for all 250 trials:



Taking the best parameters and running the simulation on the Verizon trace, we get a throughput of 4.33 Mbits/s and a 95th percentile signal delay of 186 milliseconds for a score of 23.28.

If we do the grid search on the Verizon trace directly, we find the parameters EWMA = 0.4, Target = 90, K_I = 5e-3, K_D = 1e-2, and K_P = 1e-2 for a throughput of 4.58 Mbits/s, 95th percentile signal delay of 179 ms, and a score of 25.59.

Next, we tried doing a search to find the best target delay, keeping EWMA = 0.2, K_P = 1e-2, K_I = 1e-2, and K_D = 1e-2. Here are the results:

Target	Throughput	Delay	Score
110	4.46	226	19.73
100	4.42	206	21.46
90	4.33	186	23.28
80	4.19	186	25.39
60	3.22	122	26.39
70	3.94	138	28.55

We tried running this on the TMobile trace (as a sanity check to see if we were overfitting), and we got a throughput of 10.55 Mbits/s, 95th percentile signal

delay of 198 ms, and a score of 53.28.

References

- http://www.controleng.com/single-article/fixing-pid/3975cad3f121d8df3fc0fd67660822b1.html

PIDish Controller

We found that when there were large changes in network capacity, the PID controller did not react quickly enough. In particular, when network capacity decreased rapidly, our controller didn't back off quickly enough. To account for this, we tried modifying PID so that we had two different K_D values, one for negative derivatives and one for positive derivatives.

This didn't actually make much of a difference.

PD Controller

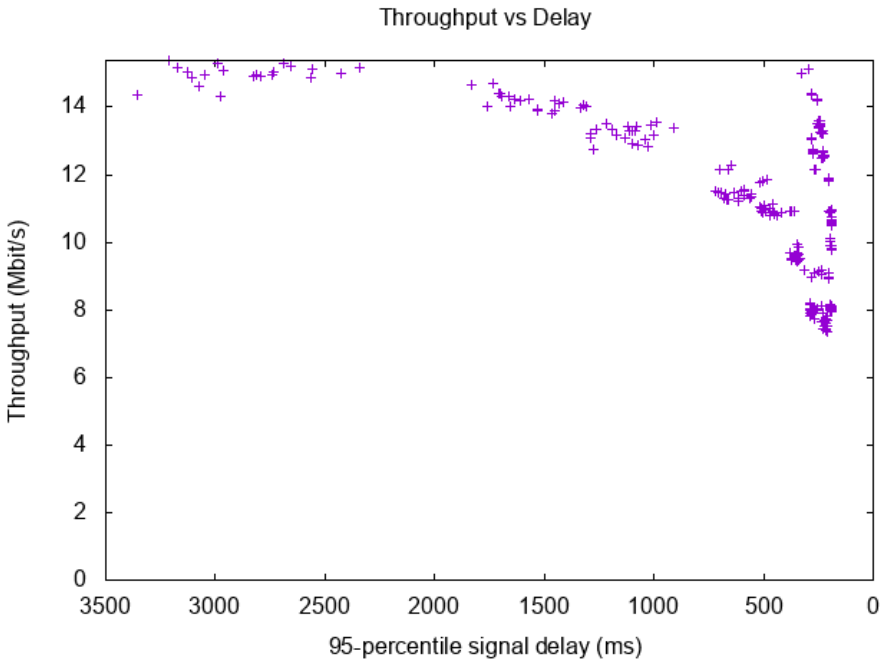
We tried using a PD controller to control changes to the window size (rather than controlling the window size directly). We did a grid search for parameters over the following space:

```
SPACE = {
    'RTT_EWMA_FACTOR': ['0.2', '0.5', '1.0'],
    'TARGET_DELAY': ['70.0', '90.0'],
    'K_P': ['1e0', '5e-1', '1e-1', '5e-2', '1e-2', '5e-3', '1e-3'],
    'K_I': ['0.0'],
    'K_D': ['1e-1', '1e-2', '1e-3', '5e-4', '1e-4', '5e-5', '1e-5'],
}
```

Here are the top results out of 294 trials, training on the TMobile dataset:

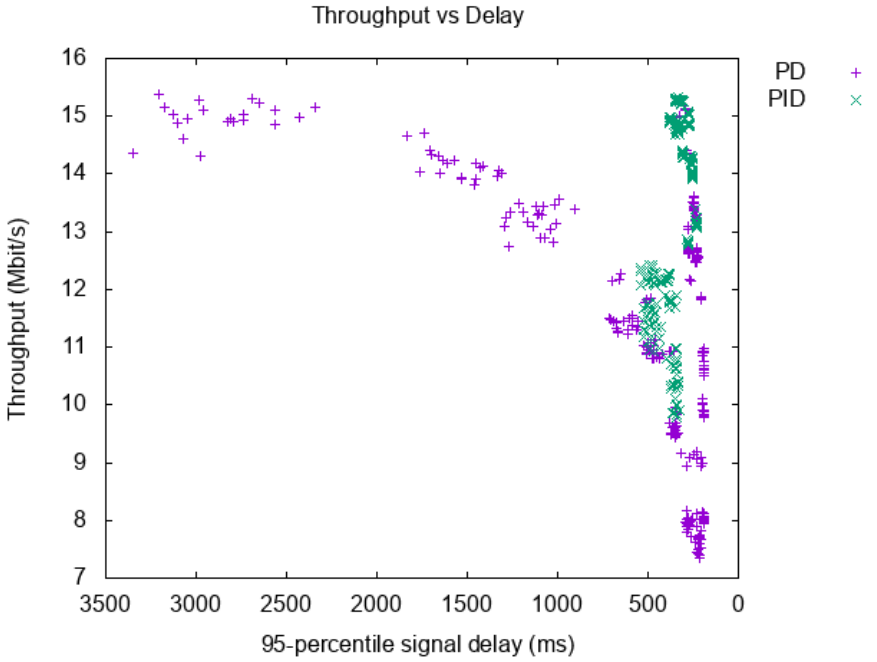
EWMA	Target	K_D	K_P	K_I	Throughput	Delay	Score
0.2	90.0	1e-2	1e-2	0.0	13.29	231	57.53
1.0	70.0	1e-3	5e-3	0.0	10.91	189	57.72
0.2	70.0	1e-3	5e-3	0.0	10.93	189	57.83
0.2	70.0	1e-2	5e-3	0.0	11.86	205	57.85
0.5	70.0	1e-2	5e-3	0.0	11.82	203	58.22
1.0	70.0	1e-2	5e-3	0.0	11.88	204	58.23

Here's a scatter plot of throughput versus delay for all 250 trials:



Using the best parameters on the Verizon dataset, we get a throughput of 4.08 Mbits/s, 95th percentile signal delay of 143 ms, and a score of 28.53.

If we compare the PID controller to the PD controller, we find the following:



PD Controller with Multiplicative Decrease

We tried augmenting our PD controller with our multiplicative decrease implementation from Part B. It didn't seem to help; the best performing runs were with higher timeouts (thus triggering the multiplicative decrease scenario very rarely.)

From the excerpt of results shown below, we observe that throughput is low in cases where the timeout that triggers a multiplicative decrease occurs, and doesn't really result in significantly lower delay. We fixed RTT_EWMA_FACTOR to 0.2, K_P to 5e-3, and K_D to 1e-2.

Timeout	Target Delay	MD Factor	Throughput	Signal Delay (ms)	Score
100	70.0	1.5	4.08	155	26.32
100	80.0	1.5	5.20	173	30.06
200	70.0	1.5	9.95	192	51.82
200	80.0	1.5	11.42	211	54.12
400	80.0	1.5	13.38	235	56.94
400	70.0	2.0	11.77	205	57.41
400	70.0	1.5	11.8	203	58.12

PD Controller for 95th Percentile Delay

As we noted earlier, our earlier PID and PD controllers were attempting to match a specific delay, but they weren't optimizing for a specific 95th percentile delay. We wrote another controller that estimated 95th percentile signal delay over a sliding window and reacted based on that. Initially, we used a simple AIMD-like scheme; later, we used a PD controller to control changes to window size.

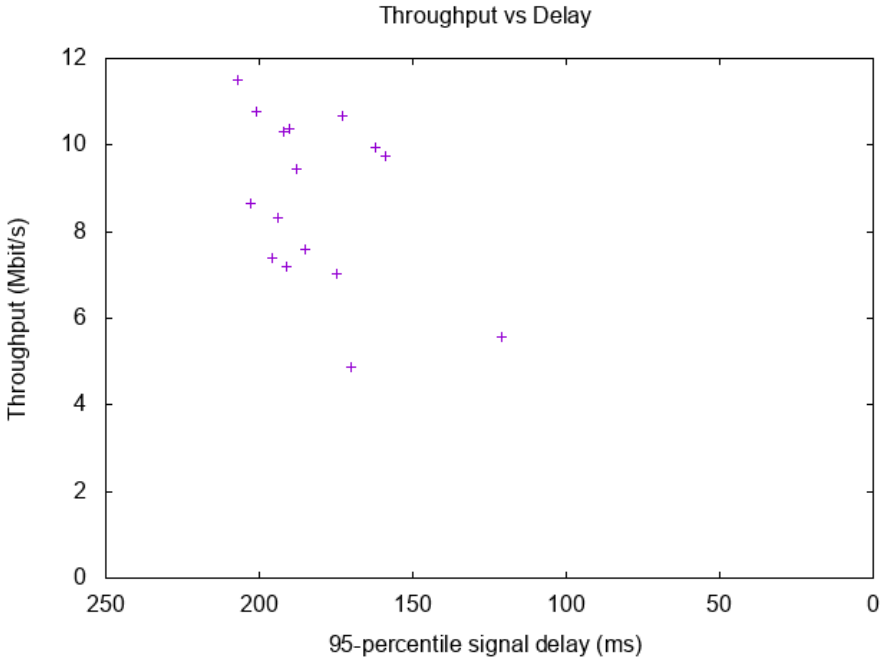
We did a small grid search over the following parameter space, training on the TMobile dataset:

```
SPACE = {
    'TARGET_95_PERCENTILE': ['80.0'],
    'K_P': ['5e-2', '1e-2'],
    'K_D': ['1e-3'],
    'CONTROL_EPOCH': ['20.0', '30.0'],
    'HISTORY_SIZE': ['3', '7'],
    'HISTORY_DECAY': ['0.8', '0.9'],
}
```

Here are the top 5 results:

Target	Control	Size	Decay	K_P	K_D	Throughput	Delay	Score
80.0	20.0	3	0.9	5e-2	1e-3	10.36	190	54.53
80.0	20.0	7	0.8	5e-2	1e-3	11.49	207	55.51
80.0	30.0	7	0.9	5e-2	1e-3	9.73	159	61.19
80.0	20.0	3	0.8	5e-2	1e-3	9.94	162	61.36
80.0	20.0	7	0.9	5e-2	1e-3	10.66	173	61.62

Here's a graph of the 16 data points:



We used the best parameters we found and ran the simulation on the Verizon dataset. We got a throughput of 3.88 Mbits/s, a 95th percentile signal delay of 152 ms, and a score of 25.53.

If we manually tweak the parameters for the Verizon dataset, we found that the best we could get was with the parameters TARGET_95_PERCENTILE = 70, K_P = 5e-2, K_D = 1e-3, CONTROL_EPOCH = 20.0, HISTORY_SIZE = 3, HISTORY_DECAY = 0.8. With these parameters, we got a throughput of 3.60 Mbits/s, a delay of 128 ms, and a score of 28.13.

Contributions

We pair programmed exercises A, B, and C. We came up with the idea for the PID controller together. Anish wrote the grid search and plotting code. We came up with the idea of the PD controller together and pair programmed it. Kate programmed the PD controller with multiplicative decrease. We came up with the idea of the PD controller for a specific 95th percentile delay together and pair programmed it.