COMP556 Project 1, Ping-Pong Client/Server

Jinlin Li
(jl288@rice.edu)

Haochen Zhang
(jz118@rice.edu)

Jiaqi He
(jh166@rice.edu)

Ye Zhou
(yz202@rice.edu)

9 februari 2023

1 Basic Implementation

We implement the client and server by event-drive based model. The server will loop forever the linked list to cope with client request. The client will send infinitely many times until server receive all data. Server also has corresponding mechanisms to receive data by reading the first two bytes data field (size). Especially when server only read one byte of data, it will wait for following term to ensure it receive correct data.
In our files, **client_num.c** and **server_num.c** are client and server source file. Simply compile them use makefile is fine.
Actually, for most of time, we will not need to send data for multiple time as the network speed. Therefore, we create a file called **stupid_client.c** which will send data by 1 byte each time and sleep for 1 secs. It turns out that the server can properly handle multiple requests of *stupid_client* simultaneously. You can compile and run it through makefile

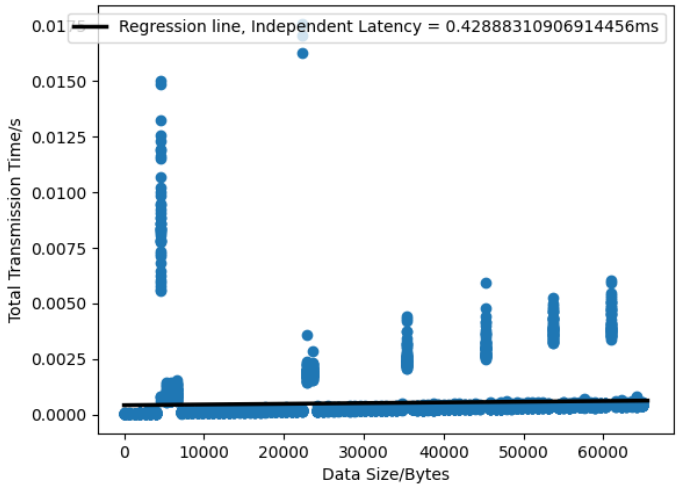
2 Latency

The client end provides the total latency. We believe the rough equation of total latency can be summarized as

$$\text{Total Latency} = k\left(\frac{\text{DataSize}}{\text{Bandwidth}}\right) + (\text{Latency that is independet with Bandwidth})$$

Therefore, we generate data according to data size and regress the data to a line. The intercept will be our result.
As it is a ping-pong server, we ask the client to do ping-pong in 100 times. Each time can give us a latency value. We do regression to all of them (100 times) (Server is ccnc-01, and client is ccnc-02)
Most of these data are correct, we remove some outliers. Personally, I believe that those

outliers can be reasoned by queuing delay or some special events of clouds. Thus, it is reasonable to remove them.
Firstly, we remove the outliers by standard deviation. , the result is 0.4289 ms, which is result_0.png. Note that y-axis is in term of second not millisecond.



Figur 1: First time remove of outliers, result_0.png

We can see that it is a line of course. But there are still many network instability points which is not removed. We use a file **test.py** to find all reasonable index and use **heuristic_remove.py** to plot our final version. (We can know that if there is no fluctuation, the last data should be the largest, therefore, we remove all data that is larger than the last data.) The result should be 0.05715ms.

Figur 2: Second time remove of outliers, result_0.png