CS655 PA1

Xingkun Yin

U44255956

**Part 1**

Logistics:

The part one implement a very simple client and server app where server echo back the message it get from client. A socket is open for connection on the port specified.

Server:

To start the server, go to pa1part1 and type in

python server.py <port number>

You can pick any port number you like. I usually pick 58989. The server will create a socket that bind to that port number and ip address of 0.0.0.0 which would bind to all interface. The server will constantly listening and reply back any message it receives.

Client:

python server.py <ip address> <port number>

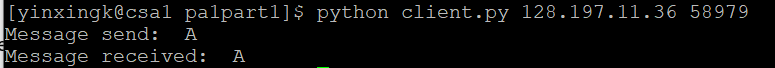
client.py takes 2 command line argument for ip and port number. It will create a socket to connect to the ip and port and send a message. It will the receive the message. Socket errors are catched.

I run server on csa2:

Text

Description automatically generated

While client is run on a different machine, namely csa1:



**Part 2**

This part consists of 3 phases, CSP, MP and CTP phase.

For all the experiments and graphs, there are 10 probes and the rtt and throughput is calculated as the average of the 10 probes.

Error handling are implemented to catch client misbehavior of sending wrong message.

For the experiment, I use my own machine as the client and CSA2 as the server. I tried to use CSA1 as the client but the rtt is too small, probably due to the close distance between CSA1 and CSA2. Server delay is set to 0.

Commands are:

python server.py 58989

python client.py 128.197.11.36 58989

Latency vs. message size, own machine client to CSA2 server

Chart, line chart

Description automatically generated

Due to the small size of the message size, bandwidth is not the main issue. Neither is queueing delay. Propagation delay probably cause the main part of the latency resulting in the volatile shape. The RTT stay roughly the same.

Throughput vs. message size

Chart, line chart

Description automatically generated

We can see a smooth grwoth in the throughput as the message size increase. This is probably due to the biggest data size here, 32K, is still not significant compare to the network capacity so the main delay is still propagation delay and therefore the RTT is roughly the same. The propagation delay is roughly the same for all message as the distance is fixed, and therefore as the message size grow, so does the throughput.

The same experiments are also conduct on a test server, I choose:

pcvm1-24.instageni.utdallas.edu 50000

python client.py pcvm1-24.instageni.utdallas.edu 50000

Latency vs. message size, pcvm1-24.instageni.utdallas.edu 50000

Chart, line chart

Description automatically generated

Throughput vs. message size

Chart, line chart

Description automatically generated

The result of the experiment towards the testing server show similar result. The RTT graph seems to have the same shape through the value on the y-axis is significantly larger. I think this is due to the propagation delay as the distance between my place to CSA2, both in Boston, is shorter than Boston to Dellas, and therefore shorter propagation delay.

The throughput graph also has similar shape as the propagation graph for my place to CSA2.

A picture containing rectangle

Description automatically generated

Chart, line chart

Description automatically generated

This is graphs using csa2 as client and pcvm1-24.instageni.utdallas.edu as server.

Server delay is set to 50, 100 and 150. The RTT seems consistent with the larger server delay, the higher RTT there is. The throughput has the same shape as expected, the more server delay, the longer the RTT is, the smaller the throughput is, but the throughput seem drop faster as the data size gets larger. We think the reasons are that system may cut messages into smaller packets and the larger the message is, the more packets there are. As the server delay is longer, there might be queueing delay as there are other people using the server as well.

Conclusion:

One possible improvement is that we can write multi-thread client and server so we can improve concurrency, use the machine more thoroughly and would reduce RTT.

Tradeoff includes