CS 550 - Spring 2022
Isaias Rivera
A20442116

P2P File Sharing - Tests

A special binary was used which does not include the interactive terminal and helps automates some of the testing. This *test peer* does the following.

- Files are randomly generated from a pool of set strings, meaning, peers will generate similar files.
- Peers also randomly delete files concurrent to actual requests.
- In addition to that, peers will also randomly generate files at runtime with the same string pool as before.
- After requesting a file and potentially making/deleting a file, the test peer will ping the server and record the response time in microseconds to an external csv file.

Due to the overwhelming number of requests to the filesystem, I had to put a delay between sequential calls to request/delete files. Otherwise, requests seemed to hang and timeout. However, calls between peers, when they happen, are still concurrent. In addition to that, one issue with the design is that the directory the program is watching will auto index files even if they are in the middle of downloading, this means files that take too long to download will actually be indexed in this half downloaded state. There are checks at the time of peer to peer connection to ensure peers actually have the file and this is only really an issue with larger files, but ultimately these requests will fail as the file no longer exists by the time a request is made. Regardless, the indexing server still processes the requests by peers and peers should be ready when this happens.

In every test, each peer aims to run 500 calls, either pinging, requesting, deleting, or creating a file, counts towards this. The binaries used are compiled as release, meaning it should run as efficient as they can, however, this also means any issues I ran into were not easily debuggable.

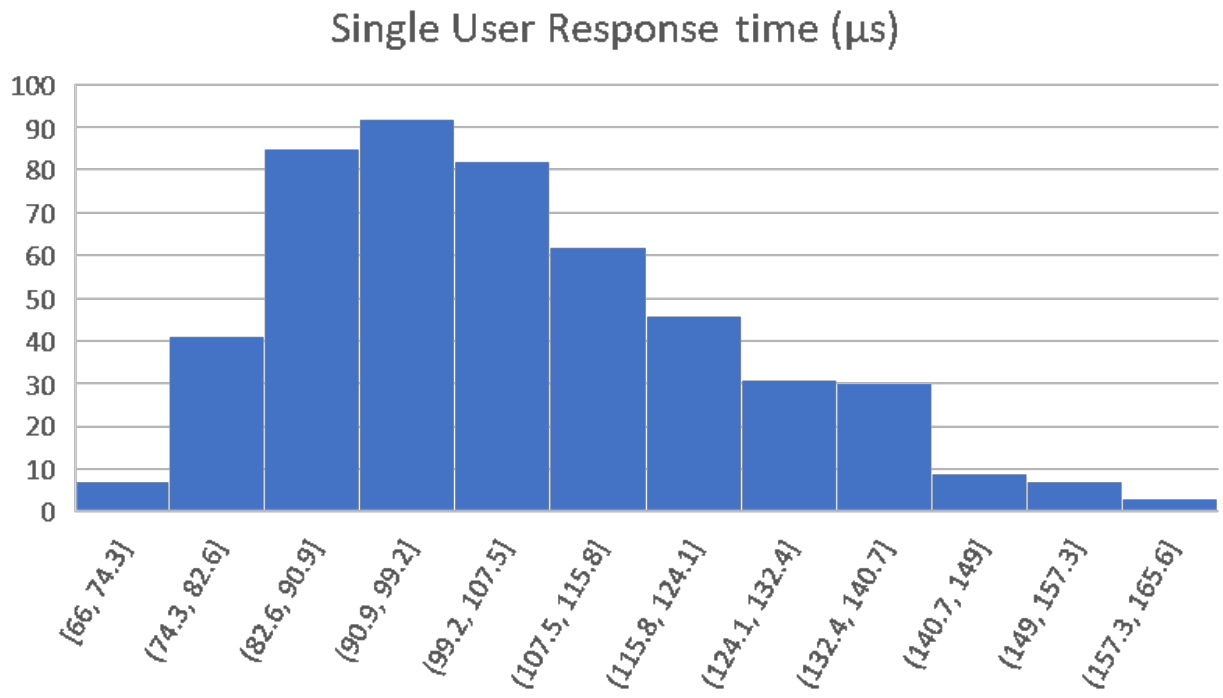
Communication between peer to peer and peer to server have timeouts, meaning, if there is sufficient traffic it is possible that requests might fail. I did not get around to implementing some method to externally catch and log these errors.

Local Stress Test

These tests were run locally on the same machine, for both peer and server.

Single User ping

This test only had a single test user connected, pinging the server 500 times. Download requests are still made, but do not count as they fail, as there is only one user connected and all of it's files are already downloaded.



Total Average = `104.2828283µs`

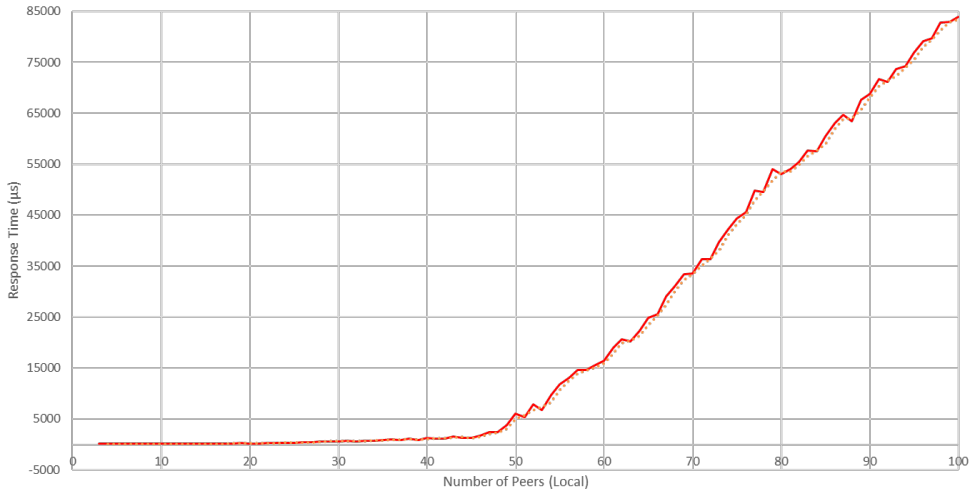
Multi User Requests

This test was automated using a python script which read the csv file generated by the test peer and then compiled it into averages. I then took these averages and graphed them in Excel.

This test concurrently ran N test peers where N ranged from 3 to 100 then, separately, 200 and 300.

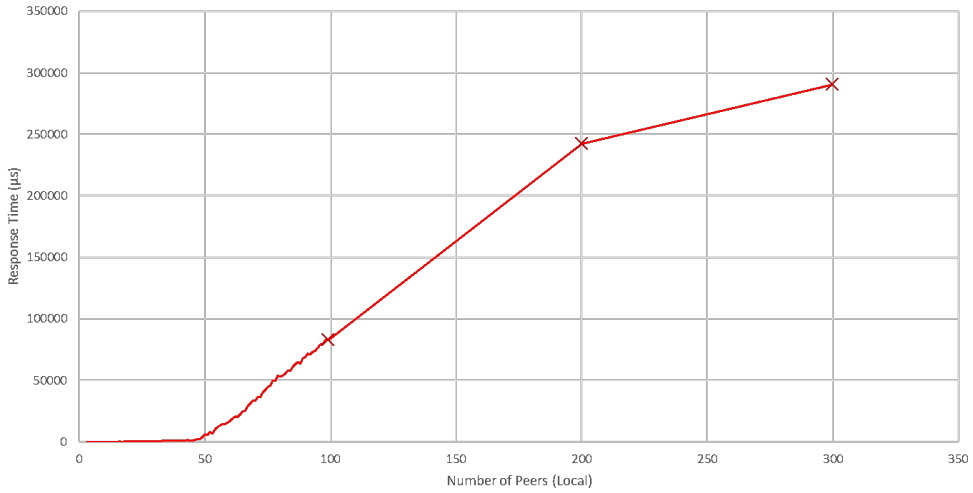
Unfortunately, beginning at around 70 peers, occasionally one of the peers would seem to hang when all the other peers have finished. And above 100 peers it got significantly worse where dozens of peers were left hanging. Because these binaries were a release version, I was unable to debug them easily and did not have time to find what the issue was. Regardless, the tests where the peers did hang were still included as only a single peer's data was lost and it did not occur very frequently below 100 peers.

Below 100 Peers There seems to be a bottle neck at around 50 peers as response time begins to steeply climb beyond this point.



At 100 peers the average was up to `83ms`.

Up to 300 Peers At 300 peers the average was up to `290ms`.



Cross Compiled function test

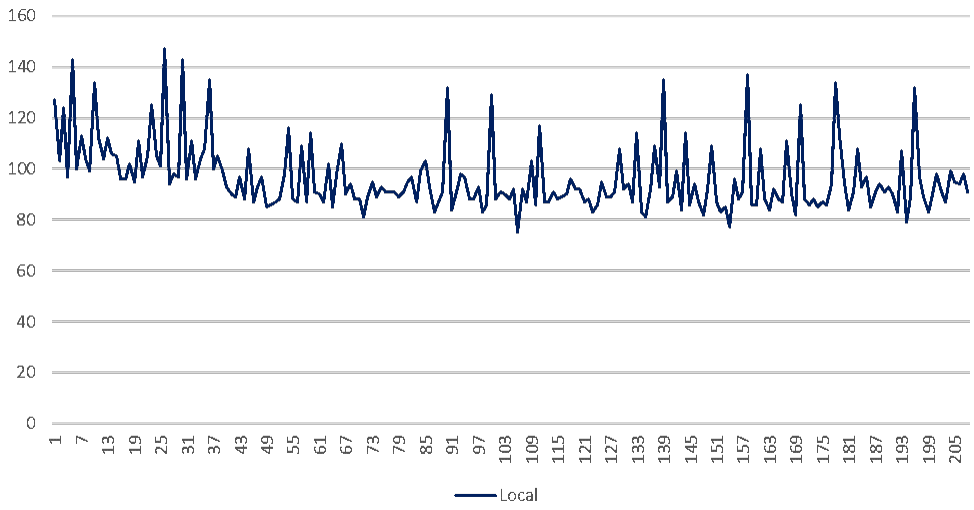
Because I did not have time to automate testing across various devices, this test mostly serves as a demonstration of the functionality working across various devices. The same code is compiled on each device but run manually, not with a script. Each device only ran one test peer and they all ran concurrently using the same server.

Device Metrics

AMD CPU - Windows 10 - x86_64

This client ran on the same machine as the server.

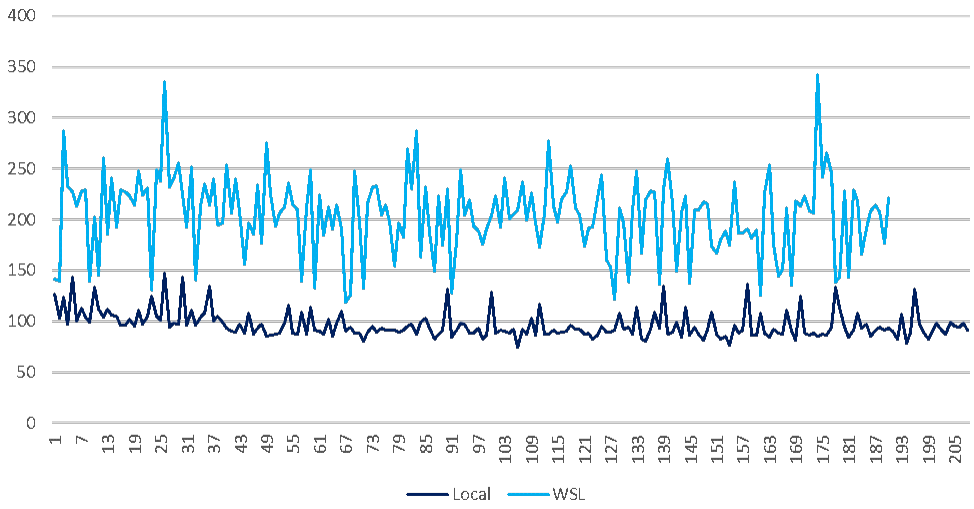
Avg Response Time = `96.38461538µs`



AMD CPU - Windows Subsystem for Linux - Debian Bookworm - x86_64

This client ran on the same machine as the server, but has additional overhead from having a compatibility layer.

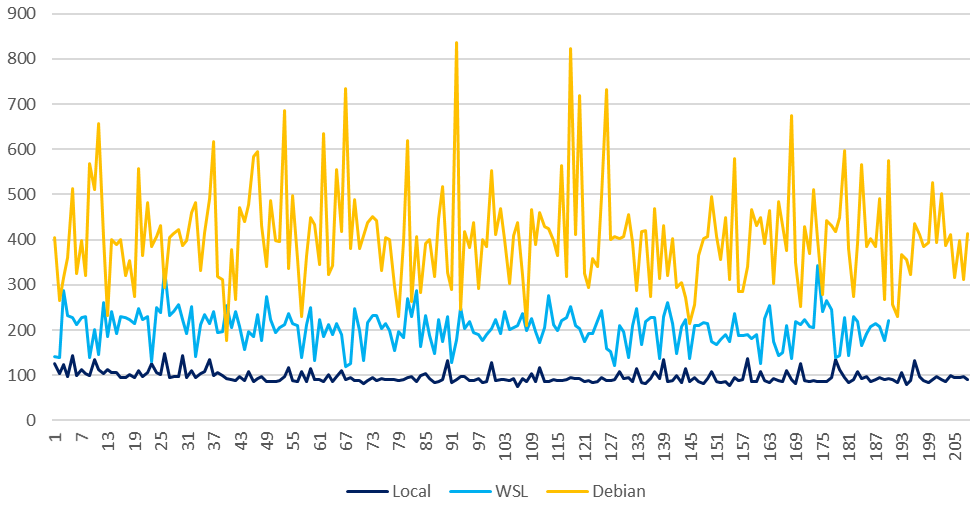
Avg Response Time = `204.8315789µs`



Intel CPU - Debian Bullseye - x86_64

This client ran on the same network as the server.

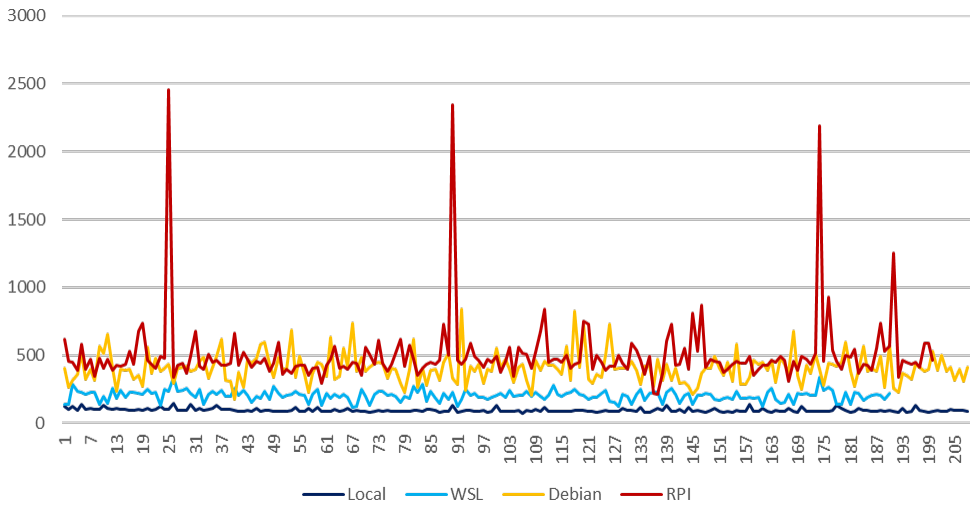
Avg Response Time = `406.9759615µs`



Raspberry Pi 4 - Debian Bullseye - ARM

This client ran on the same network as the server. The spikes in response time seem like they might be following a pattern.

Avg Response Time = `503.93µs`



Android w/ Termux - Debian Bullseye - ARM

This client ran on the same network as the server but through WIFI.

Avg Response Time = `4629.076923µs`

Took the longest with a maximum of `45ms`. The spikes in response time here also seem like they might be following a pattern.