CS 550 - Spring 2022
Isaias Rivera
A20442116

P2P File Sharing - Tests

A special binary is used which does not include the interactive terminal. Files are randomly generated from a pool of set strings, meaning, peers will have similar files. Peers also randomly delete files concurrent to the requests. 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 seem to hang and timeout. However, calls between peers, when they happen, are still concurrent. Additionally, 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 for peer to peer connection, but ultimately these requests will fail as the file no longer exists by the time a request is made.

It is not often but clients will sometimes give the error that they were unable to deregister a file but the indexing server actually did deregister the file.

In every test, each peer aims to run 500 calls, either pinging, requesting, or deleting counts towards this. The binaries used are compiled as release.

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 do not believe this occurred, at least in the tests shown here. Regardless, I did not get around to implementing some method to externally catch and log these errors.

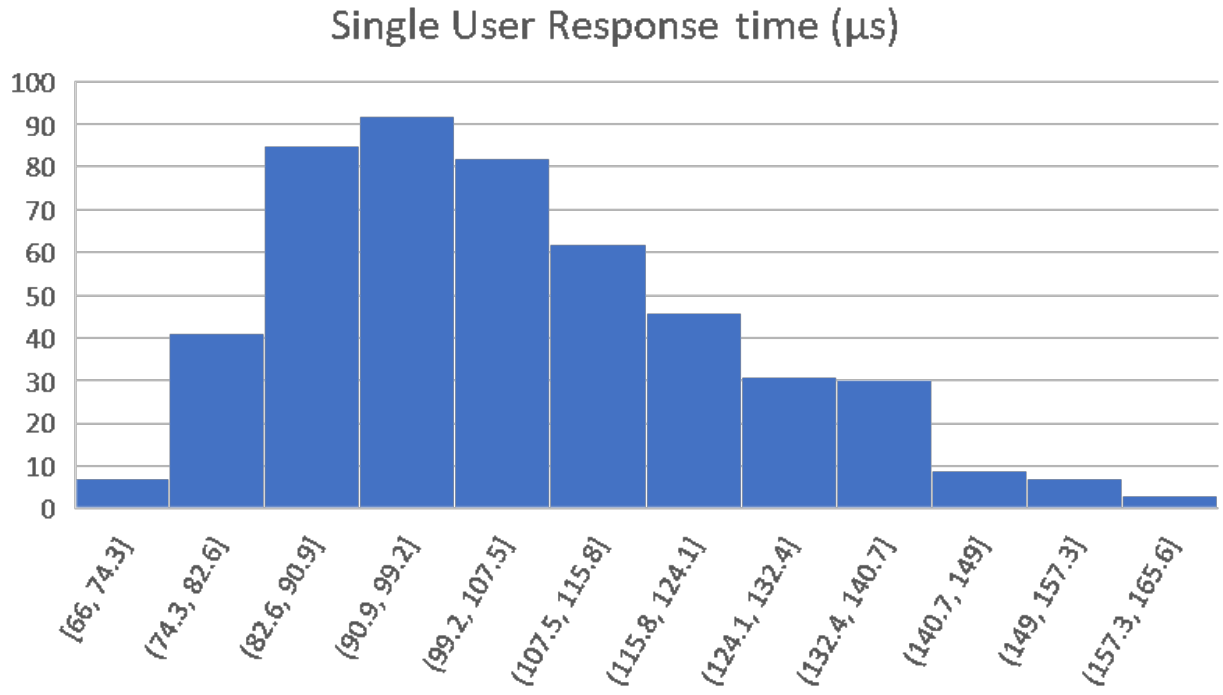
These tests were automated using a python script, which reads the csv file generated py the test programs and compiling it into averages. I then took these averages and graphed them in Excel.

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.

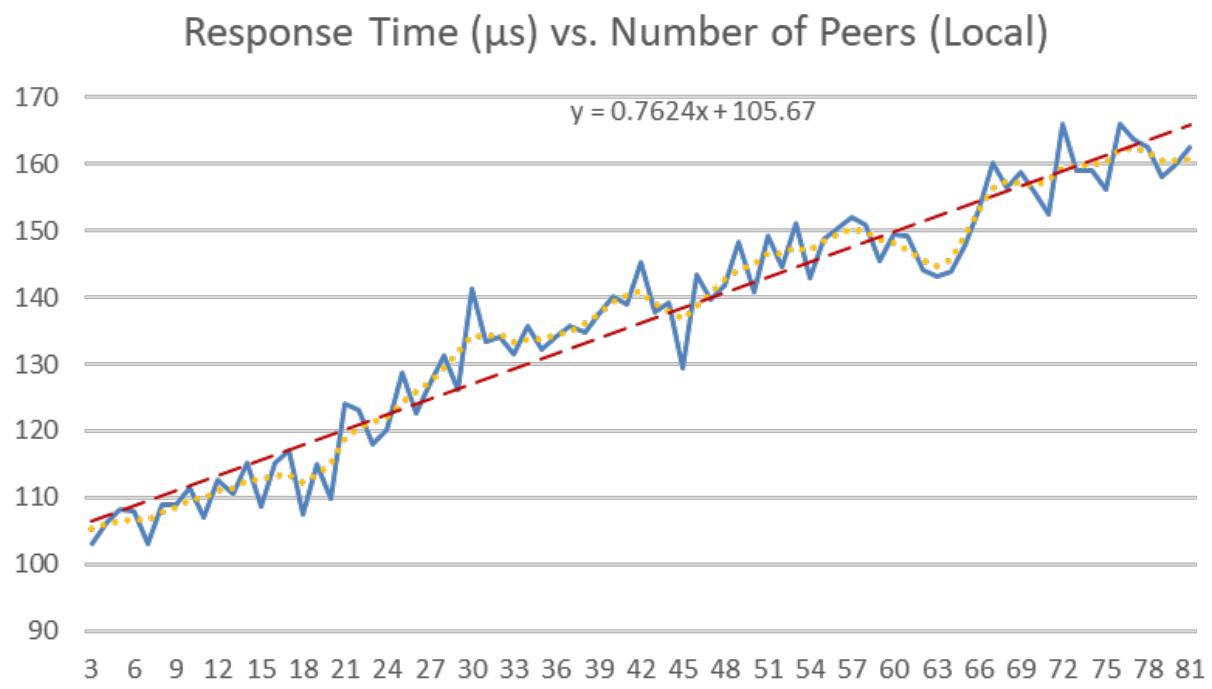


Total Average = `104.2828283µs`

Multi User Requests

This test concurrently ran N test peers where N ranged from 3 to 81.

Unfortunately, at 82 peers 3 of the peers seemed to hang. Because these binaries were a release version, I could not debug them easily and did not have time to find what the issue was. However, the growth in response time seems to be linear.



Min Time = `103.0071735µs @ 7 Peers`

Max Time = `166.0021375µs @ 72 peers`

Cross Compiled function test

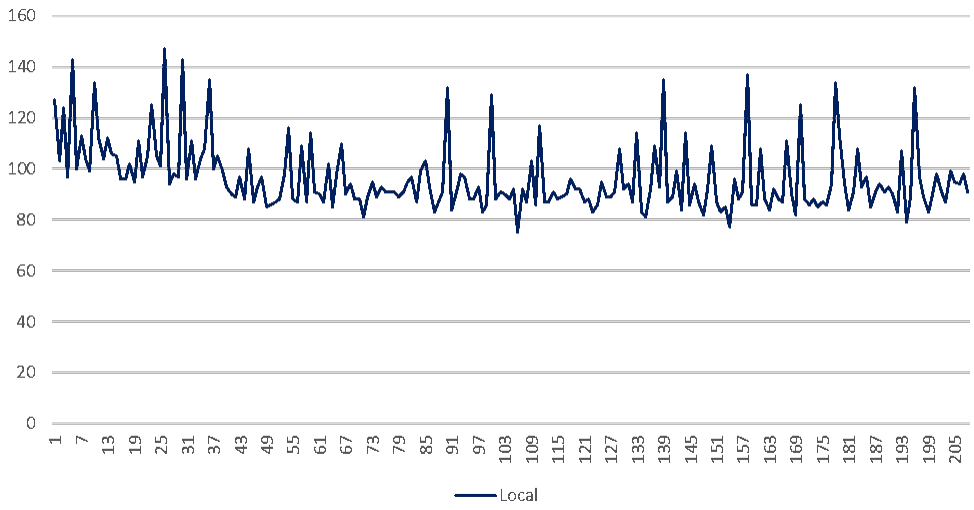
Because I did not have time to automate testing across various devices, this test mostly serves as a demonstration of the functionality working across various devices. The same code is compiled on each device but run manually, not with a script. Each device only ran one test peer and they all ran concurrently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

Device Metrics

AMD CPU - Windows 10 - x86_64

This client ran on the same machine as the server.

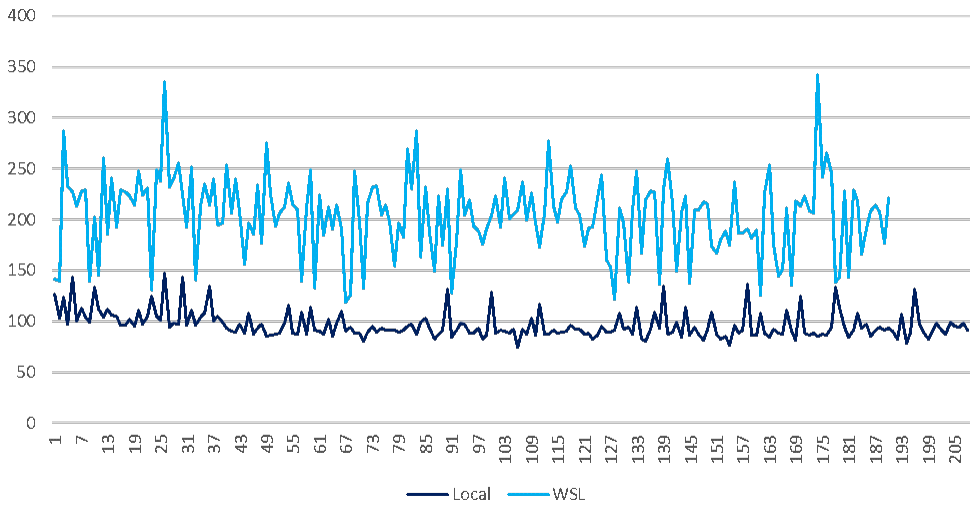
Avg Response Time = `96.38461538µs`



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.

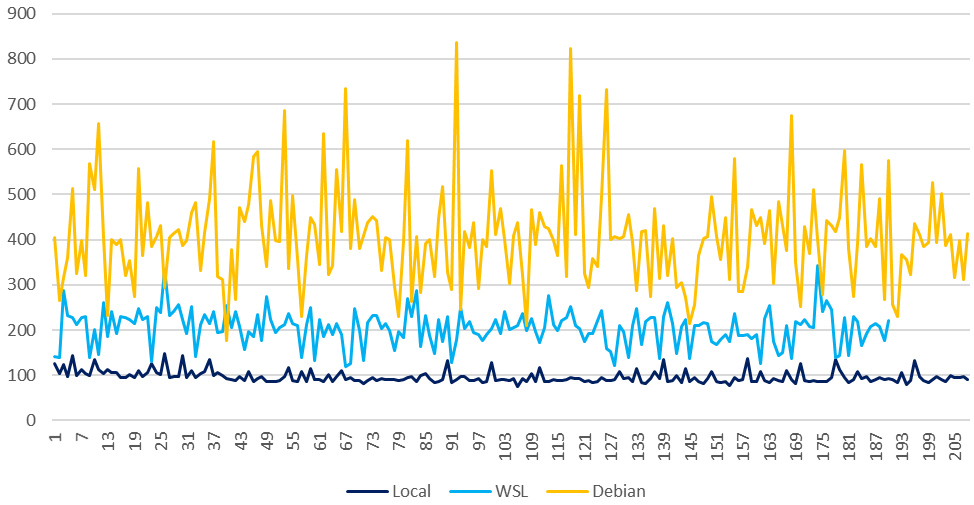
Avg Response Time = `204.8315789µs`



Intel CPU - Debian Bullseye - x86_64

This client ran on the same network as the server.

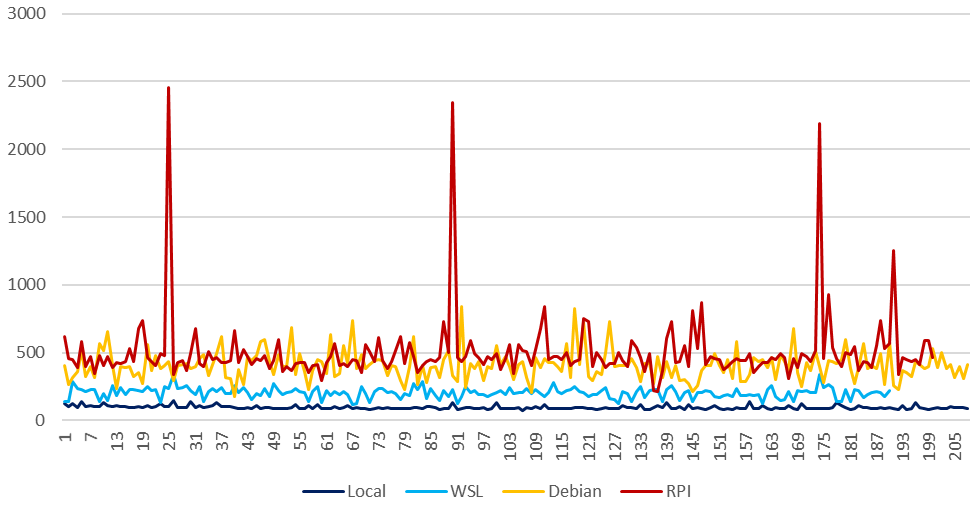
Avg Response Time = `406.9759615µs`



Raspberry Pi 4 - Debian Bullseye - ARM

This client ran on the same network as the server.

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, by far, the longest with a maximum of 45ms.