CS 550 - Spring 2022
Isaias Rivera
A20442116

P2P File Sharing - Super-peers - Tests

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

- Files are randomly generated from a pool of set strings, meaning, peers will often 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 200 calls, only the actual requesting of a file from super-peers counts towards this. For each client, the response time for each request of this type is stored to a csv. 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 super-peer indexing 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 Tests

These tests were run locally on the same windows machine, for both peer and super-peer.

10 Super-peers and 19 Leaf-nodes

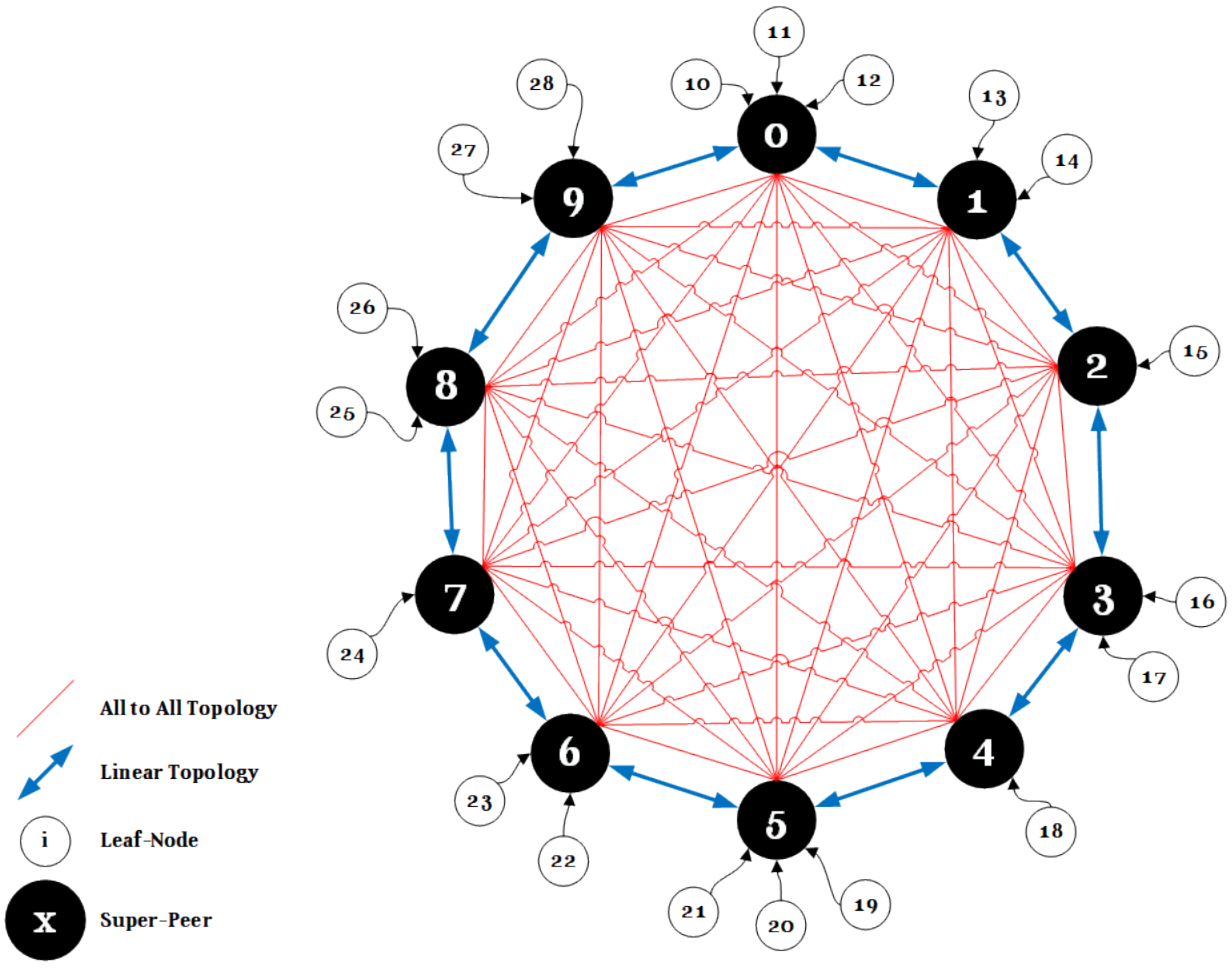
This test was automated using a python script which read the csv file generated by the test peer and then compiled it into averages. I then took these averages and graphed them in Excel.

This test concurrently ran 29 test peers where 10 of them were super peers. Only N number of these peers are actively making requests, where N ranged from 1 to 29.

This test is performed for both the linear and all to all topology.

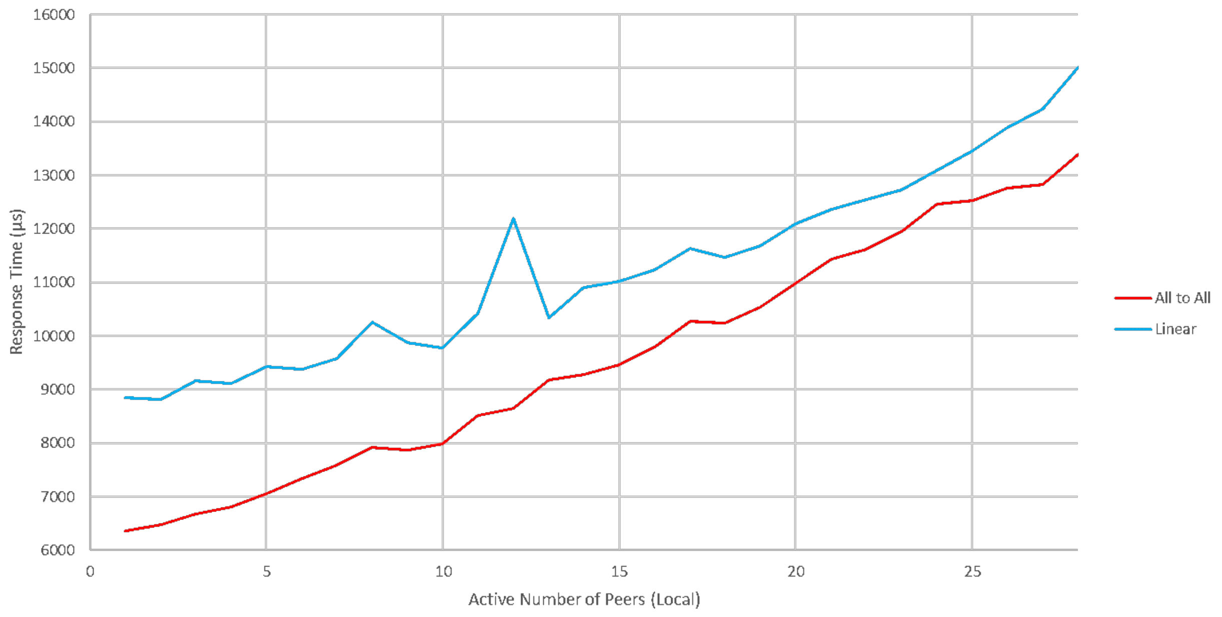
Topology setup

This is a graphic which visualizes how each client is setup using the static configuration.



Results

This graph plots both the times for the linear and all to all topologies.



Linear topology is probably not actually used in practice as it requires for all supers peers to either keep track of active requests, keep track of their neighbors, and return results through all the supers peers. There is simply a lot of extra overhead when using a linear topology. The All to All topology is not only more straight forward to implement but also takes advantage of parallel operation. On average, the response time for both topologies grows similarly. However, the linear topology has more overhead, meaning it is always slower than the all to all topology. Because these tests were all run on the same machine, I imagine this difference would be a magnitude higher if it were actually deployed.