James Gojit, ID#015785846

Spring 2021 - CECS 327 Sec 05

Peer-to-Peer File Share: Write-Up

This project was a very fruitful experience. As someone who has not had any previous with distributed systems and implementing anything related to networks, this project gave me the opportunity to explore just very little of quite a useful subject field. With the freedom to go about implementing the requirements for this project, I learned a lot.

To start, the concept of threads was crucial into being able to get all the features of a single program running as both the client-end and server-end of an application. Little of my knowledge goes into operating systems, so doing some research into context switching and thread synchronization allowed me to have some idea as to implement different parts of the application. Using Java, basic multi-threading was very easy to accomplish. Although I most definitely could have designed the project such that information could be passed across concurrent threads in a more efficient and fail-safe manner, I got it far enough so that the program can run threads for handling the client’s presence on the network, listeners and handlers for active TCP connections, and file transfer.

Referring to the client’s presence on the local network, I first considered having a distributed hash table for creating the network overlay. I looked into options such as implementing Kademlia and Chord. With my current understanding of some DHT architectures, much of it involved distributing the files to certain nodes. In my case, I wanted to have nodes all have the most recently updated copies of all files rather than having individual peers be responsible for some number of files. I also considered having a super peer that is solely responsible for holding a mapping of clients on the network and providing means for nodes to connect to the super peer and fetch some number of those mappings. With that, I found that implementing a UDP broadcast to be much easier to comprehend with my understanding.

Now that I was able to locate peers on the network, I needed to establish connections. For file transfer, TCP sockets were the easiest method for me to understand and try to implement. I did consider using Java’s Remote Method Invocation (RMI) library to perform these functions. Using RMI would allow me to perform file transfers across the network without having to integrate application specific protocols for communication. In the end, I continued to use TCP sockets where a client runs a ServerSocket thread for handling incoming Socket requests from other peers. With that, learning how to send file data over the network was a Google search away. The hardest part of this project for me was deciding how to implement bi-directional synchronization using TCP sockets. I went with a command-based approach, in which some client thread would interact with a peer’s server thread, passing commands back and forth to perform desired functions such as requesting, sending, and uniting files.

In closing, there is much more I would like to do with this project. My program demo executes these functions on manual trigger. If I were to continue improving my work, I would most definitely have a watcher thread examining the local directory for updates. In such cases, I would like peers to synchronize. On the topic of synchronization, there are holes in my implementation. One such error is when two separate clients are simultaneously contacting a third client and trying to overwrite the same file on the third client. Otherwise, this project was very flexible. I most definitely appreciate that such flexibility allowed me to learn so much more.