**CS 550 - Spring 2022**
**Isaias Rivera**
**A20442116**

# P2P File Sharing - Design

## Initial Constraints

This project code is portable to various devices. This greatly affected the design in terms of what libraries or functionality I had available to me. It runs on Linux, Linux Arm, and Windows (Mac untested). However, I did put my focus on Windows as that is where I did my programming, meaning there might be a bug or two that I missed on Unix systems.

This project is written in C++20 meaning it requires a more modern compiler and meant to run on newer devices. This does, however, allow me to use more modern C++ features.

## Indexing Files

### Communication

The communication between peer and server is purely through RPC. This is because the data being sent between peer and server is often of a known size and the data being sent isn't very large, in most cases.

### Hashing

The main idea behind this design for indexing is that each file should be identified by it's hash first, name second. Meaning, each file on the indexing server has a unique hash, as files do, but multiple files can have the same name. This might not be the best way of doing it, however, at least for now, this is how I implemented it and gives me leeway for other indexing implementations.

### "Database"

The indexing server's "database" is just two maps. One that associates a unique id to each peer and another that associates a hash to a file.

Each peer also references all the files that it has and each file references each peer that has it. This circular referencing allows for easier searching of either files or peers and can also help ensure files/peers are properly added/removed.

### Watch Folder

Peers are not allowed to directly interact with the register or deregister functions, instead, there is a "watch folder" that is defined by passing the program the correct argument. This directory is for both receiving and sending files to other peers, but also for files that we care to send to the indexing server. Changes to this directory (added, deleted, or modified a file) will automaticlly update with their respective function (register, deregister, or both when file is modified).

### Searching

Currently, searching is very basic. Peers can search the indexing server by passing a query to the `search` function. It returns files whose names contain the query. Because multiple files can have the same name, it also returns the hash of the file, which is used to actually choose the file to download.

Peers also have the option of listing every available file, however, this is more meant for testing as it would not make sense to actually have something like this in the real world.

### Requesting

Peers can request a file to download to their "watch folder" by passing a hash of a file to the `request` function. This will both list available peers from the indexing server and select the first available one. It will then begin the process of requesting the specific file from this peer.

## Peer to Peer

### Communication

Peers connect to each other using only sockets. This allows it to be faster than RPC but it also means that communication is at a lower level.

### State Machine

The protocol for communication is known between both peers and is setup as a state machine. The following sections are each state for communication. For this section, *client* refers to the peer making a request and *server* refers to the peer receiving the request.

**Connection** Client sends request for connection the the server known to have the correct file. This step is handled by separate threads that are either accepting or sending requests.

**Confirm ID** The server sends it's ID to the client to confirm it connected to the correct server. If so, the client sends the file hash, otherwise it disconnects.

**Find File** The server receives the hash and attempts to find the file locally. If found, it will return the file size and begin waiting for confirmation to start streaming data. If not found, disconnect.

**Allocate Space** As the name suggests, this is where the client would pre-allocate space for the incoming file, unfortunately, I did not get to implementing this part. Regardless, this is where the client opens a new file to write to. It then send confirmation that it can begin streaming data.

**Streaming** Server streams chunks of the file to the client and the client writes these chunks onto the open file. Streaming is finished once the client no longer receives any data.

### Error handling

When an error occurs with a peer, ID mismatch, hash mismatch, file not found, etc. The client peer attempts to request the next peer in the list it initially received from the indexing server. An error is printed if it fails with all peers.

### Console interaction

Only the peer is really meant to interact with the program directly, they have various functions exposed for them to use.

- `ping`
  - Ping the server to check it's response time
- `list`
  - List all the files on the indexing server
- `search [query]`
  - Search for the query as a substring the in the name of all the files on the indexing server
- `request [hash]`
  - Request a specific file, given the hash
- `register` & `deregister`
  - Not allowed but still listed
- `q` || `quit` || `exit`
  - Stop the Interactive Console

### Program arguments

#### Peer

Client is passed on start up it's unique ID, the address for the server, and port to listen to for p2p, and the "watch folder" directory path

#### Indexing Server

Server is only passed the port that it should listen on.

### Improvements

There are a lot of things that I would like to have implemented but did not have time to, the following is a list of some of those things.

- Peer offline detection
  - Server would ping each peer periodically to check that they are on, if they are not it would delete every entry for that peer
- Server offline detection
  - If the indexing server were to go offline the peer would attempt to reconnect and re-register all of it's files
- Ignore files that are being actively modified
  - Files that are located in the "watch folder" will hash and send the file information to the indexing server each time it is edited. I found this is especially bad when editing a file on a Unix device, as each edit even without saving would ping the indexing server. This also includes files that are being downloaded to the "watch folder"
- True concurrent file indexing
  - Currently the "Database" is just two unordered maps, there should be a way to allow fully concurrent read/write to this data structure, currently a mutex is used to ensure no race conditions.
- Communication between peers is rather crude
  - It would be best to use RPC alongside sockets instead of just sockets when it comes to peers
- Peer file verification
  - Peer should be able to verify the size of the file it is going to download not just the hash
  - Peer should also check the file hash matches the indexing server listing after downloading from a peer
  - Peer should also pre-allocate space for the file when it receives the file size
- Multi peer download
  - Especially for larger files, add capability to download separate chunks from multiple peers
- Smarter peer selection
  - Instead of picking the first available peer, ping each peer, or only a couple if there are many, and choose the fastest one
- Better search function
  - Use a proper search function instead of just checking each file for a substring when searching for files