

Peer-to-Peer Distributed File Sharing System Report

1. Introduction

This report outlines the implementation logic of the Peer-to-Peer Distributed File Sharing System developed in C++. The system allows users to share and download files within groups they belong to. It supports parallel downloading from multiple peers, a fallback multi-tracker system, and a custom piece selection algorithm.

2. Implementation Logic

2.1 Tracker

- The tracker maintains information about clients and the files they share.
- It synchronizes data across multiple trackers to ensure consistency.
- Key data structures:
 - `unordered_map<string, user>`: Stores user credentials.
 - `unordered_map<string, group>`: Stores group information.
 - `unordered_map<string, vector<string>>`: Manages group join requests.
- The tracker handles various client commands like `create_user`, `login`, `create_group`, `join_group`, etc.
- It communicates with clients over TCP sockets, using multithreading to handle multiple clients concurrently.

2.2 Client

- The client allows users to perform actions such as creating accounts, logging in, creating/joining groups, uploading/downloading files, etc.
- It establishes a listening socket to serve file chunks to other peers.
- The client communicates with the tracker to get information about available files and peers sharing them.
- The `get_file()` function implements the download logic:
 - Uses a round-robin piece selection algorithm to distribute chunk requests among available peers.
 - Creates an output file of the correct size filled with zeros.
 - Uses `lseek()` to write received chunks to the correct position in the file.
 - Verifies the integrity of the downloaded file by comparing hashes.

2.3 Piece Selection Algorithm

- Implemented a round-robin algorithm for piece selection.
- Chunks are distributed among peers by assigning each chunk to a peer based on the chunk number modulo the number of peers.
- This ensures that chunks are downloaded from multiple peers, enhancing download speed and network utilization.

2.4 File Integrity Verification

- SHA1 hashing is used to verify the integrity of both individual chunks and the complete file.
- After downloading, the client computes the hash of the downloaded file and compares it with the hash provided by the tracker.

3. Features Implemented

- **User Account Management:** Create and manage user accounts with authentication.
- **Group Management:** Create groups, join/leave groups, and handle join requests.
- **File Sharing:** Upload files to groups and list available files.
- **Parallel Downloading:** Download files by fetching chunks from multiple peers simultaneously.
- **Custom Piece Selection:** Implemented a round-robin algorithm to select which peer to download each chunk from.
- **File Integrity:** Ensured file integrity through SHA1 hash comparisons.
- **Multithreading:** Used multithreading to handle multiple client connections and peer communications.
- **Error Handling:** Added comprehensive error handling for socket operations and file I/O.

4. Assumptions Made

- At least one tracker is always online.
- Peers have the necessary permissions to read the files they are sharing.
- Network connections between clients and trackers/peers are reliable.
- The number of chunks is correctly calculated based on the file size and chunk size (512 KB).

5. References

- Socket Programming:
- Beej's Guide to Network Programming: <https://beej.us/guide/bgnet/>
- Multithreading in C++:
- C++ Reference - `<thread>`:
<https://en.cppreference.com/w/cpp/thread>
- File I/O and Filesystem:
- C++ Reference - `<filesystem>`:
<https://en.cppreference.com/w/cpp/filesystem>
- SHA1 Hashing with OpenSSL:
- OpenSSL SHA1 Documentation: <https://www.openssl.org/docs/man1.1.1/man3/SHA1.html>
- Error Handling and Exception Safety:
 - "Effective C++" by Scott Meyers

6. Conclusion

The implemented Peer-to-Peer Distributed File Sharing System fulfills the required functionalities, including synchronized trackers, client group management, and parallel file downloading with a custom piece selection algorithm. Through the use of multithreading and proper synchronization, the system can handle multiple clients and peers efficiently.