AOS Assignment 3
Peer-to-Peer Distributed File Sharing System
Monsoon 2025

International Institute of Information Technology, Hyderabad

Assignment Deadline: September 30, 2025 11:59 PM

1 Assignment Overview

1.1 Deadlines

- **Interim Submission:** September 18, 2025, 11:59:00 PM
- **Final Submission:** September 30, 2025, 11:59:00 PM

1.2 System Description

You will build a distributed file sharing system that allows users to form groups and share files within those groups. The system operates on a peer-to-peer model where files are not stored on central servers but are distributed across participating clients. When a user wants to download a file, the system coordinates multiple peers to provide different pieces of the file simultaneously, enabling faster downloads and better resource utilization.

The system consists of tracker servers that maintain metadata about users, groups, and file locations, while the actual file data is transferred directly between clients. Users must authenticate themselves, join or create groups, and can then share files with other members of their groups. The system supports concurrent downloads, where multiple pieces of different files can be downloaded simultaneously, and implements integrity checking to ensure downloaded files are not corrupted.

Two tracker servers work together to provide redundancy and maintain consistent information about the entire network. When one tracker receives updates about new users, groups, or shared files, this information must be synchronized with other trackers to ensure the system provides accurate and up-to-date information regardless of which tracker a client connects to.

2 System Components

2.1 Tracker System

The tracker system consists of exactly two tracker servers that maintain synchronized state information. Trackers handle user registration and authentication, manage group memberships and permissions, store metadata about shared files including their locations and integrity hashes, and provide peer discovery services for file downloads. The tracker system must ensure that all information remains consistent across both trackers, and the system should continue operating as long as at least one tracker remains online.

2.2 Client System

Client applications allow users to interact with the file sharing network. Clients handle user authentication and session management, provide interfaces for group creation and management, manage file upload and download operations, coordinate with multiple peers for parallel downloads, and maintain local file piece information for sharing with other peers. Each client acts both as a downloader (requesting files) and as a seeder (providing files to others).

3 Technical Requirements

3.1 File Management

Files in the system are divided into pieces of exactly 512KB each, with the final piece potentially being smaller if the total file size is not evenly divisible by 512KB. Each piece and the complete file must have SHA1 hash verification to ensure integrity. The system must efficiently handle large files up to 1GB in size without excessive memory usage. When pieces are downloaded, they must be immediately verified against their expected hashes, and corrupted pieces should be re-downloaded from different peers.

3.2 Network Communication

All network communication uses TCP sockets with custom protocols designed by the implementer. The system requires three types of communication: tracker-to-client for user management and metadata exchange, client-to-client for direct file piece transfers, and tracker-to-tracker for synchronization. The protocol design must handle partial message transmissions, network failures, and provide appropriate error codes and status messages.

3.3 Concurrency and Performance

The system must support multiple simultaneous downloads per client, with different files being downloaded concurrently and individual files being downloaded from multiple peers simultaneously. All shared data structures must be thread-safe, and the implementation should efficiently manage system resources including threads, memory, and file descriptors. The design should balance performance with resource constraints.

4 Core Functionalities

4.1 User and Group Management

Users interact with the system through the following commands:

`create_user <user_id> <password>` - Register a new user account
`login <user_id> <password>` - Authenticate and start a session
`create_group <group_id>` - Create a new group (user becomes owner)
`join_group <group_id>` - Request to join an existing group
`leave_group <group_id>` - Leave a group you're a member of
`list_groups` - Display all available groups in the system
`list_requests <group_id>` - Show pending join requests (owner only)
`accept_request <group_id> <user_id>` - Accept a join request (owner only)
`logout` - End current session and stop sharing files

4.2 File Operations

File sharing and downloading operations include:

`upload_file <group_id> <file_path>` - Share a file with a group
`list_files <group_id>` - Show all files available in a group
`download_file <group_id> <file_name> <destination_path>` - Download a file from the group
`show_downloads` - Display current download progress
`stop_share <group_id> <file_name>` - Stop sharing a specific file

4.3 System Execution

To run the system components:

`./tracker tracker_info.txt tracker_no` - Start a tracker server
`./client <IP>:<PORT> tracker_info.txt` - Start a client application
`quit` - Shutdown tracker (used within tracker console)

5 Implementation Constraints

5.1 Programming Requirements

The implementation must be written entirely in C or C++ using only standard system calls and libraries. High-level libraries such as filesystem libraries, external torrent implementations, or database libraries are strictly prohibited. Functions like system(), exec family functions, and popen() are not allowed. The code must compile cleanly with standard gcc/g++ compilers and should include appropriate Makefiles for easy compilation.

5.2 Code Quality and Structure

Code must be well-organized into logical modules with clear separation of concerns. Comprehensive error handling is required for all operations including network failures, file system errors, and memory allocation failures. All dynamically allocated memory must be properly freed, and all open file descriptors and sockets must be closed appropriately. Thread safety must be ensured for all shared data structures, and the code should include clear comments explaining complex algorithms and design decisions.

5.3 Network Protocol Design

Implementers must design their own network communication protocols for all interactions. The protocol should include appropriate message headers, error codes, and status indicators. Message integrity should be ensured, and the protocol should handle partial transmissions gracefully. The design should be efficient and extensible while remaining simple enough to implement reliably.

6 File Integrity and Piece Management

The system ensures file integrity through SHA1 hashing at both the piece level and complete file level. When a file is shared, the system calculates SHA1 hashes for each 512KB piece as well as for the entire file. These hashes are stored and distributed through the tracker system. During downloads, each received piece is immediately verified against its expected hash, and any corrupted pieces are discarded and re-requested from different peers.

The piece selection algorithm for downloads is left to the implementer's design. Strategies might include sequential downloading, random piece selection, or more sophisticated algorithms that consider piece rarity or peer reliability. The chosen algorithm should ensure efficient utilization of available peers while avoiding unnecessary duplicate downloads.

7 Tracker Synchronization

The two-tracker system requires a synchronization mechanism to maintain consistent state across both servers. When any tracker receives updates about new users, group changes, or file sharing information, this data must be propagated to other online trackers. The synchronization approach is left to the implementer, but the system must ensure that clients receive accurate and current information regardless of which tracker they connect to.

The synchronization mechanism should handle scenarios where trackers are temporarily disconnected from each other, and should ensure that the system remains functional as long as at least one tracker is operational. Recovery procedures should be in place for when a tracker comes back online after being disconnected.

8 Download Progress and Status

The system must provide clear feedback about ongoing downloads using the following status format:

`[C] [group_id] filename` - Indicates a completed download.

9 Testing and Validation

The implemented system will be tested with various file sizes ranging from small files under 512KB to large files approaching 1GB. Testing will include concurrent operations with up to 3 clients and 2 trackers running simultaneously. The system must handle network interruptions gracefully, including tracker failures, peer disconnections during downloads, and network partitions that temporarily separate system components.

Testing scenarios will include different file types such as text files, binary executables, images, and compressed archives. The system should maintain data integrity across all file types and sizes while providing reasonable performance for typical use cases.

10 Submission Requirements

10.1 File Structure

Submit your implementation as a ZIP file named `<Roll_Number>_A3.zip` containing:

- `tracker/` directory with all tracker source files
- `client/` directory with all client source files
- `README.md` file with comprehensive documentation
- Optional `Makefile` in each directory for compilation

10.2 Documentation Requirements

The README.md file must include detailed compilation and execution instructions, architectural overview of your design, explanation of key algorithms implemented, data structures chosen and their rationale, network protocol design and message formats, assumptions made during implementation, list of implemented features and any limitations, and comprehensive testing procedures.

Additionally, prepare a separate technical report explaining your implementation approach, synchronization algorithm design, piece selection strategy, protocol design rationale, challenges encountered and solutions implemented, and references to any external resources consulted during development.

11 Evaluation Criteria

The assignment will be evaluated based on functional correctness of all required operations, efficiency and robustness in handling error conditions and edge cases, and understanding demonstrated during the viva examination.

Particular attention will be paid to the correctness of the tracker synchronization mechanism, effectiveness of the multi-peer download implementation, proper handling of file integrity verification, and overall system reliability.

12 Important Guidelines

All code submissions will be thoroughly checked for plagiarism against current and previous year submissions as well as AI-generated content. Code that fails to compile will receive zero marks, and implementations with segmentation faults during testing will be heavily penalized. Memory leaks will be detected and penalized accordingly.

Students should focus on creating robust, well-designed implementations rather than trying to implement every possible feature. A smaller set of features implemented correctly and reliably will score better than a larger set of features with reliability issues.

The viva examination will focus on understanding of the implemented design, ability to explain architectural decisions, knowledge of the algorithms used, and capability to modify or extend the implementation. Students should be prepared to discuss trade-offs in their design choices and demonstrate thorough understanding of their code.