Lab 1: Peer-to-Peer File Sharing Simulation

Anupama Srikanthan (axs7911)
Surya Prasad Senthilkumaran(skp6318)

CSE 514: Computer Networks
The Pennsylvania State University
October 17, 2025

Contents

1 System Description

The system emulates a sample Peer-to-Peer (P2P) communication system between multiple peers and one central tracking server. Unlike the traditional client-server system, the clients (referred to as Peers) contribute to exchange of data in the system, while one central server tracks the status of the peers in the system at any time.

1.1 Central Server

The server acts as a central snapshot or "tracker" of the system at any point in time. Its primary responsibilities include:

- **File Registry:** Maintaining a master list of all files available for download on the network.
- **Peer Tracking:** For each file, the server keeps track of the peers (identified by IP and port) that have specific chunks of any given file.

The protocols that govern the communication between the peers and the tracking server are present in the `models/` directory.

1.2 Peer

The peer is the client application that both shares and downloads files. The former is called a `Seed` while the latter is called a `Leech` in popular applications like BitTorrent, so they are referenced as such here. The peer implementation (`p2p-peer/main.py`) can be started in one of two modes (using the `--mode` argument):

- **Seed Mode:** The peer connects to the server and registers all files located in its sharing directory (`--share-dir`). It then listens for incoming connections from other peers (leechs) to upload file chunks.
- **Leech Mode:** The peer connects to the server and presents a menu to the user. The user can list available files from the server, select a file to download, or exit.

1.3 File Transfer Process

The file transfer process follows the system description:

1. A leech requests the "File Locations" for a specific file from the server.
2. The server responds with a list of peers and the chunks they each possess.
3. The leech's `download_file` function plans the download, implementing a "rarest first" chunk selection strategy.

4. The leech spawns multiple threads to download different chunks simultaneously from multiple peers.
5. As the leech successfully downloads and verifies a chunk, it registers this new chunk with the server using a `CHUNK_REGISTER_REQUEST`. This allows the leech to become a seed for that chunk for other peers.

2 Protocol Specifications

The protocol is built on TCP sockets and uses JSON-encoded messages for communication between the server and peers. Pydantic models are used on both the server and Peer side to define and validate the structure of these messages (see `models/` directory).

This implementation directly follows the minimal message list suggested in the lab guideline.

2.1 Server-Peer Protocol

Register Request Message: `REGISTER_REQUEST`
Model: `RegisterRequest` (from `models/peer_request_models.py`)
Payload: Contains Peer Endpoint and List of Files to Register.

Register Reply Model: `FileRegisterReply` (from `models/server_response_models.py`)
Payload: A response stating status and message indicating if registration was successful.

File List Request Message: `FILE_LIST_REQUEST`
Model: `FileListRequest` (from `models/peer_request_models.py`)
Payload: No request body.

File List Reply Model: `FileListReply` (from `models/server_response_models.py`)
Payload: The number of files, list of file names and lengths.

File Locations Request Message: `FILE_LOCATIONS_REQUEST`
Model: `FileLocationsRequest` (from `models/peer_request_models.py`)
Payload: The name of the requested file.

File Locations Reply Model: `FileLocationsReply` (from `models/server_response_models.py`)
Payload: A dictionary mapping peer IDs to the list of chunk indices they possess.

Chunk Register Request Message: `CHUNK_REGISTER_REQUEST`
Model: `ChunkRegisterRequest` (from `models/peer_request_models.py`)
Payload: The file-name, chunk-indicator and endpoint of the new Seed.

Chunk Register Reply Model: `ChunkRegisterReply` (from `models/server_response_models.py`)
Payload: A status and message indicating if the chunk was registered.

2.2 Peer-to-Peer Protocol (Chunk Transfer)

File Chunk Request Model: `FileChunkRequest` (from `models/peer_request_models.py`)
Payload: A JSON object sent from the leech to the seed, specifying the file and chunk-indicator requested.

File Chunk Reply Model: `FileChunkReply` (from `models/peer_request_models.py`)
This is not a single model but a stream of bytes. The seed's `handle_chunk_request` function first sends a custom JSON header containing the chunk's size and its SHA-256 hash. It then sends the raw chunk data as a byte stream.

3 Program Structure

The repository is organized into three main Python packages: `p2p-server`, `p2p-peer`, and `models`.

`p2p-server/`

- `main.py`: The main entry point for the server. It simply instantiates and starts the `P2PServer`.
- `server_socket.py`: Contains the `P2PServer` class. This class binds the server socket, listens for connections, and spawns a new thread (`handle_client`) for each peer. The `handle_client` method contains the core logic for parsing requests and updating the `file_database` registry.

`p2p-peer/`

- `main.py`: The main entry point for the peer. It uses `argparse` to read command-line arguments (like `--mode`, `--share-dir`, `--peer-port`). This fulfills the interface requirement to specify shared files via command line.
- `client_chunk_process.py`: Contains the `Peer` class, which holds all peer-side logic. This includes:
  - `connect_to_server()`: Manages the connection to the central server.
  - `register_files_with_server()`: Handles the "seed" mode logic.
  - `start_upload_server()`: Starts the peer's own listening socket to handle upload requests from other peers.
  - `handle_chunk_request()`: The seed-side logic for reading a file chunk and sending it (with a hash header) to a leech.
  - `leech_menu()`: The main interface for "leech" mode, fulfilling the requirement to list files and choose a download.
  - `download_file()`: Implements the logic for fetching file locations, planning the "rarest first" download, and starting parallel download threads.
  - `leech_chunk()`: The leech-side logic for downloading a chunk, verifying its hash, and saving it to a `.piece` file.

  - `assemble_file()`: Reconstructs the final file from its pieces after all downloads are complete.

`models/` This package contains Pydantic models for data validation, separating the protocol definition from the network logic.

- `common_models.py`
- `peer_request_models.py`
- `server_response_models.py`

This is done so that the network protocols can be extended to/inherited as interfaces.

4 Features & Implementation Status

This section describes which parts of the lab specification work and which do not.

4.1 Working Features

All basic and group requirements from the lab guideline are satisfied (in the scope of this project)

- **Multiple Connections:** The server (`P2PServer`) and the peer's upload server (`start_upload_server`) both use `threading` to handle multiple simultaneous connections.
- **Parallel Downloading:** The `download_file` method spawns a separate `threading.Thread` for each chunk download, allowing chunks to be downloaded from multiple peers in parallel.
- **Chunk Download Completion:** Upon finishing a chunk download and verifying its hash, the `leech_chunk` function calls `register_chunk_with_server`, which sends a `CHUNK_REGISTER_REQUEST` to the server. This makes the leech a new source for that chunk.
- **Integrity Check:** The `handle_chunk_request` (seed) function calculates and sends a SHA-256 hash of the chunk data. The `leech_chunk` (leech) function verifies this hash upon receipt. If the hashes do not match, the leech prints an error and discards the piece, fulfilling the integrity check requirement.
- **Chunk Selection (Rarest First):** The `download_file` function explicitly implements the "rarest first" group requirement. It counts the availability of each chunk across all peers and sorts the download list to prioritize the rarest chunks.

- **Interface:** The peer uses `argparse` for command-line setup (specifying share directory) and provides a text-based menu for listing and downloading files. Download progress for each chunk is displayed using `tqdm`, fulfilling the "view progress" requirement.

4.2 Known Issues & Limitations

- **[Selective File Select]:** *Currently, all the files in a directory are registered with the server.*
- **[Stale Data on Crash]:** *e.g., A peer abruptly disconnecting might leave stale data on the server. A heartbeat or timeout mechanism is not implemented. There is no synchronization mechanism either so if a peer deletes it's file, the server does not get notified with it.*

5 Compilation and Execution

This section provides the system description and commands for running the program, as required by the submission guidelines.

5.1 Language and Dependencies

The project is implemented in **Python**. All required dependencies are listed in `requirements.txt`.

A `Makefile` is provided to automate installation and execution. To install all dependencies into a Python virtual environment (`env-p2p-comm-sim`), run:

```
make install
```

5.2 Running the System

All commands should be run from the root directory of the project.

1. Start the Server: Open a terminal and run:

```
make run-server
```

The server will bind to `0.0.0.0:7777` (as configured in `p2p-server/main.py`).

2. Start a Peer (seed): This peer will register files from `./seeding-directory` and listen for uploads on port 8001 (by default).

```
# Run with default settings (PORT=8001, SHARE_DIR=./seeding-directory)
make run-seed

# Or, override the settings:
make run-seed PORT_SEED=9001 SHARE_DIR_SEED=./my-files
```

3. Start a Peer (leech): This peer will save completed files to `./complete-files` and listen for uploads on port 8002 (by default).

```
# Run with default settings (PORT=8002, SHARE_DIR=./complete-files)
make run-leech

# Or, override the settings:
make run-leech PORT_LEECH=9002 SHARE_DIR_LEECH=./my-downloads
```

6 Sample Output

This section provides sample output of the program, as requested in the guidelines.

6.1 Server Output

```
Server: Listening on 0.0.0.0:7777
Server: Connected to ('127.0.0.1', 51000)
Server: Handling REGISTER_REQUEST from 127.0.0.1:51000
Server: Connected to ('127.0.0.1', 51001)
Server: Handling FILE_LIST_REQUEST from 127.0.0.1:51001
Server: Handling FILE_LOCATIONS_REQUEST from 127.0.0.1:51001
Server: Handling CHUNK_REGISTER_REQUEST from 127.0.0.1:51001
Server: Handling CHUNK_REGISTER_REQUEST from 127.0.0.1:51001
...
```

6.2 seed (Peer 1) Output

```
Peer: Connected to server at ('127.0.0.1', 7777)
Peer, register_files_with_server: Registering 3 files with server.
Peer, register_files_with_server: Server response - Status: success, ...
Peer: Establishing new seed
Peer: Seed listening on 127.0.0.1:9001 for uploads.
Peer: Connected to leech at ('127.0.0.1', 51005)
Peer: handle_chunk_request: Received request for chunk 5 of bork.png from ...
Peer: handle_chunk_request: Showcase of Integrity Check: chunk_hash: a1b2c3d4...
Peer: Connected to leech at ('127.0.0.1', 51006)
Peer: handle_chunk_request: Received request for chunk 2 of bork.png from ...
```

6.3 leech (Peer 2) Output

```
Peer: Connected to server at ('127.0.0.1', 7777)
Peer: Establishing new leech
```

```
Peer: Seed listening on 127.0.0.1:9002 for uploads.

--- Leech Menu ---
1. List available files
2. Download a file
3. Exit
Enter your choice: 1
- berk.jpg (Size: 153023 bytes)
- birk.jpg (Size: 13916 bytes)
- bork.png (Size: 1221588 bytes)

--- Leech Menu ---
1. List available files
2. Download a file
3. Exit
Enter your choice: 2
Enter the name of the file to download: bork.png

Peer: leech_chunk: Downloading chunk 0/9 from 127.0.0.1:9001.: 100%|...| 120K/120K
Peer: leech_chunk: Piece 0 of bork.png received successfully from 127.0.0.1:9001.
Peer: create_piece_file: Saved piece 0 of bork.png to complete-files/bork.png_pieces/...
... (10 chunks download) ...

Peer: Assembling file bork.png from pieces.
File bork.png assembled successfully in ./seeding-directory.
Peer, register_file_with_server: Registering file bork.png with server.
Peer, register_file_with_server: Server response - Status: success, ...

--- Leech Menu ---
1. List available files
2. Download a file
3. Exit
Enter your choice: 3
```

A Source Code

Source code is provided in a Github Repository, fulfilling the final submission requirement. A clean version history is also displayed (though major contributions were made in a single laptop at one point). All major functions within `client_chunk_process.py` and `server_socket.py` include docstrings or comments explaining their purpose.