P2P File Sharing Project

By Eden Eitani

207762246

For Course 20588

December 2024

Table of Contents

1. **Introduction**
   1.1 What is P2P?
   1.2 What is BitTorrent?
2. **Prerequisites**
   2.1 Required Software
   2.2 Installation Process
3. **Architecture**
   3.1 Tracker
   3.2 Client (Seeder and Leecher)
   3.3 Protocol
   3.4 Torrent
   3.5 Communication
4. **Flow of Operation**
   4.1 Uploading a File
   4.2 Downloading a File
   4.3 Seeding
5. **Communication Protocol**
   5.1 Operation Codes
   5.2 Return Codes
   5.3 Payload Fields
   5.4 Example Message Flows
6. **System Structure**
   6.1 Services
   6.2 Network
   6.3 Input and Output Folders
7. **Running the System**
   7.1 Default Setup
   7.2 Interacting with Peers
   7.3 Running Tests
8. **Error Handling and Retry Mechanism**
9. **File Handling**
   9.1 File Splitting
   9.2 Chunk Hashing
   9.3 Encoding and Decoding
10. **Conclusion and Future Work**

1. Introduction

Sharing files over the internet has become a common part of our lives, and **peer-to-peer (P2P) file sharing** is one of the most innovative ways to do it. Unlike traditional methods where a central server manages everything, P2P allows users to share files directly. This approach is not only faster and more efficient but also ensures there's no single point of failure, making it incredibly reliable.

In this project, we're building a P2P file-sharing system where files are split into smaller chunks. These chunks can be downloaded from several users at the same time, speeding up the process and making better use of everyone's internet connection. It's a simple yet powerful way to share files.

1.1 What is P2P?

P2P (peer-to-peer) networking is all about sharing resources directly between users, without needing a middleman. Think of it as a group of friends sharing items among themselves rather than having to go to a central store. Each participant in a P2P network called a "peer," can act both as a supplier and a receiver. This makes the system very flexible and robust.

Key features of P2P include:

- **No Central Authority**: There's no single server running the show, so if one part of the network goes down, the rest keeps running.
- **Resource Sharing**: Everyone pitches in—whether it's bandwidth, storage, or processing power.
- **Scalability**: The more people join, the better it works. More peers mean more resources to share.

1.2 What is BitTorrent?

BitTorrent is one of the most popular ways to use P2P technology, especially for sharing large files. Instead of downloading a whole file from one person, BitTorrent splits the file into smaller pieces, which can be shared among many people. This way, users can upload and download parts of the file simultaneously, making the process faster and more efficient.

Key concepts in BitTorrent:

- **Seeders and Leechers**: If you've downloaded the whole file, you're a "seeder" and can help others get it. If you're still downloading, you're a "leecher" but can still share the parts you've already received.
- **Tracker** : A tracker helps coordinate who's sharing what, and which files are registered to the system.

In this project, I've implemented aspects of both BitTorrent and P2P file sharing to create a fast and reliable system for sharing files.

2. Prerequisites:
    a. The system works with Python and docker and can be installed on every supported system (either Mac, Windows, or Linux)
    b. Docker
    c. Python3.8+

3. **Architecture**
   a. Tracker
      i. Manages the available torrents
      ii. Tracks active peers
   b. Client
      i. Represents both seeder and leecher
      ii. Handles uploading and downloading file chunks
      iii. Communicates with the tracker and other peers in the system.
      iv. Once a peer connects as a leecher and downloads a file - it becomes a seeder.
      v. A peer can only seed one file at a time but can continue downloading available torrents.
   c. Protocol
      i. Defines the communication standards in the system.
   d. Torrent
      i. Represents a single torrent file with its metadata
      ii. Handles corresponding peers connected to certain torrents (as seeder or leecher)
   e. Communication
      i. TCP communication for peer to peer communication, chunk transfer and tracker to peer communication.
      ii. Ensures data integrity and completeness.
   f. Multithreading
      i. The system uses threads using asyncio for it’s operation.
      ii. Once a seeder uploads a file it creates a listening server for download requests in another thread, so it can continue its operation and download files.

iii. The tracker also uses threads to accept multiple requests concurrently.

4. **Flow of operation**
    a. Upload a file
        i. A peer registers to the tracker, uploads a torrent and becomes the initial seeder of the torrent
    b. Download a file
        i. A peer registers to the tracker, asks for a list of available torrents and requests the torrent to download.
        ii. The torrent will be downloaded in chunks from all available seeders.
    c. Seeding
        i. Once the download is completed the peer becomes a seeder of the file as well and registers to the tracker as a new seeder of the torrent.
        ii. A peer can only seed one file, but can continue to download other files available.

5. **Communication protocol**
   a. We've defined a protocol of communication between server to peer and peer to peer, also operation codes in the system.
   b. **Operation Codes**
      i. **Peer-Tracker Operations (`PeerServerOperation`)**:
         1. `GET_LIST`: Request a list of available torrents from the tracker.
         2. `GET_TORRENT`: Fetch metadata for a specific torrent.
         3. `START_SEED`: Notify the tracker that the peer is seeding a file.
         4. `STOP_SEED`: Notify the tracker that the peer is stopping the seeding process.
         5. `UPLOAD_FILE`: Register a new file for sharing and initiate seeding.
      ii. **Peer-Peer Operations (`PeerOperation`)**:
         1. `GET_PEERS`: Retrieve a list of peers sharing a specific file.
         2. `GET_CHUNK`: Request a specific chunk of the file from another peer.
         3. `STATUS_INTERESTED`: Notify a peer of interest in downloading from them.
         4. `STATUS_CHOKED`/`STATUS_UNCHOKED`: Indicate the availability or unavailability of chunks to other peers.
   b. **Return Codes**
      1. **Success Codes**:
         a. `SUCCESS (200)`: Operation completed successfully.
         b. `FINISHED_DOWNLOAD (201)`: Peer has finished downloading the file.

   c. `FINISHED_SEEDING (202)`: Peer has finished seeding the file.

2. **Error Codes**:
   a. `BAD_REQUEST (400)`: Malformed or invalid request.
   b. `TORRENT_DOES_NOT_EXIST (411)`: Requested torrent ID does not exist.
   c. `NO_AVAILABLE_TORRENTS (410)`: No torrents available for sharing.
   d. `FAIL (450)`: Generic failure.

3. **Payload Fields**
   A. **General Fields**:
      a. `OP_CODE`: Specifies the operation being performed.
      b. `RET_CODE`: Indicates the return code of the operation.
      c. `IP_ADDRESS` and `PORT`: Network details for peer-to-peer communication.
   B. **Torrent-Specific Fields**:
      a. `TORRENT_ID`: Unique identifier for a torrent.
      b. `FILE_NAME`: Name of the file being shared.
      c. `NUM_OF_CHUNKS`: Number of chunks the file is divided into.
      d. `SEEDER_LIST`/`LEECHER_LIST`: Lists of peers seeding or downloading the file.
   C. **Chunk-Specific Fields**:
      a. `CHUNK_IDX`: Index of the requested or shared chunk.
      b. `CHUNK_DATA`: Actual data of the chunk.

5.1 Example Communication Flows

1. Peer Requests File List

- **Request**:
  - `OP_CODE`: `GET_LIST`
  - `IP_ADDRESS`: Peer's IP
  - `PORT`: Peer's port
- **Response**:
  - `RET_CODE`: `SUCCESS`
  - `TORRENT_LIST`: List of available torrents.

2. Peer Downloads a File

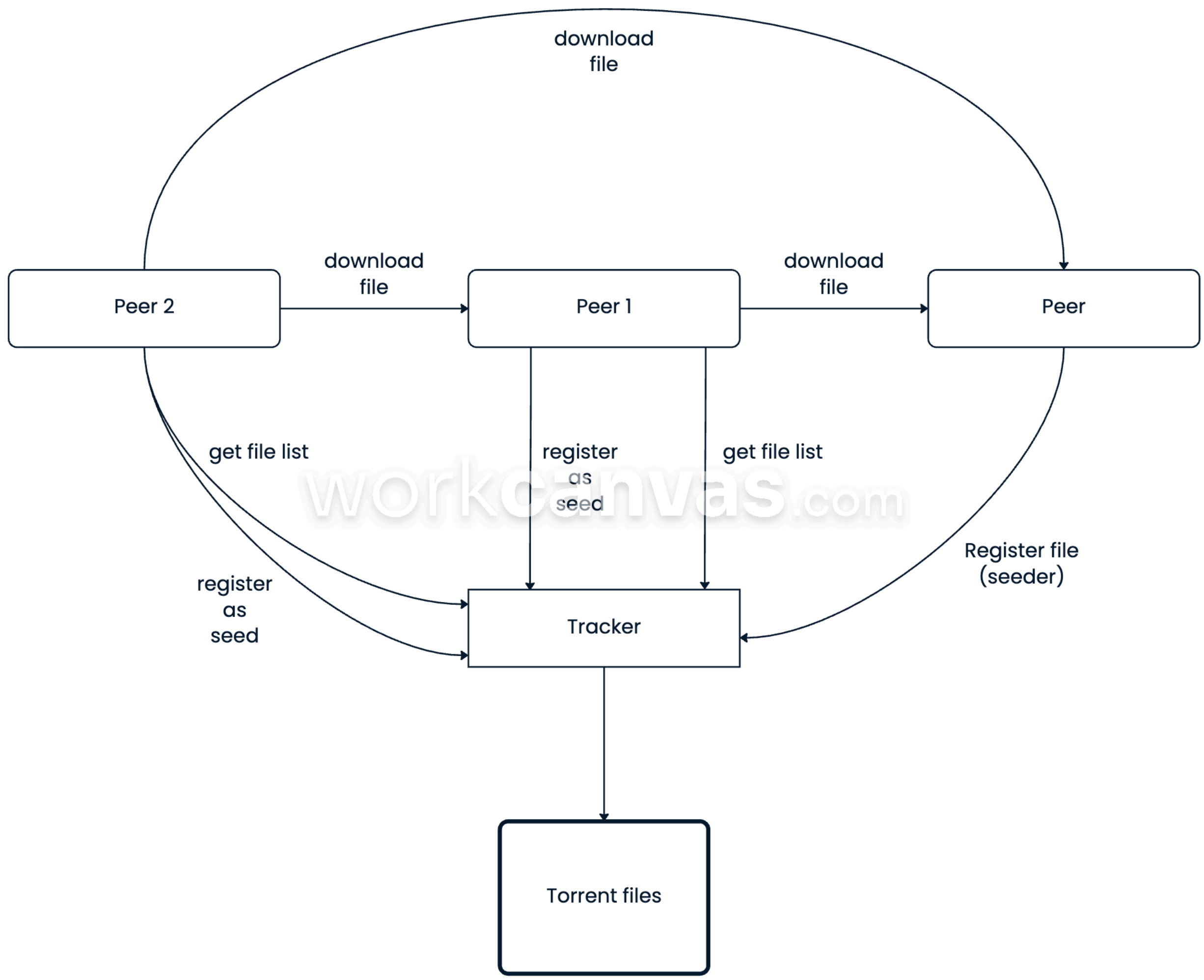
- **Request Metadata**:
  - `OP_CODE`: `GET_TORRENT`
  - `TORRENT_ID`: Torrent ID of the desired file.
- **Request Chunks**:
  - `OP_CODE`: `GET_CHUNK`
  - `CHUNK_IDX`: Index of the desired chunk.

3. Tracker Updates Seeding Status

- **Request**:
  - `OP_CODE`: `START_SEED`
  - `TORRENT_ID`: Torrent ID of the file being seeded.
  - `PEER_ID`: ID of the peer initiating the seed.
- **Response**:

- `RET_CODE`: `SUCCESS`.

5.2. Architecture Diagram:



5.3. Possible Flows:

1. Single Peer Download

1. Peer requests the chunk list from the tracker.
2. Tracker provides the chunk metadata and seeder details.
3. Peer sequentially requests chunks using `GET_CHUNK`.
4. Seeder responds with `CHUNK_DATA` for each request.
5. Peer assembles chunks into a complete file.

2. Multi-Peer Download (Multiple seeders)

1. Peer requests the chunk list and seeder information from the tracker.
2. Peer splits chunk requests across available seeders.
3. Each seeder responds with `CHUNK_DATA`.
4. Peer assembles all received chunks.
5. Peer notifies the tracker of completion and transitions to a seeder.

3. Parallel Peer Downloads (Multiple Leechers)

1. Each peer retrieves chunk and seeder details from the tracker.
2. Peers independently request different chunks to minimize duplicate requests.
3. Seeder(s) handle concurrent chunk requests.
4. Each peer completes the download and informs the tracker.

4. Failure and Recovery

1. Peer requests chunks from seeders.
2. If a request fails, the peer retries with another seeder or the same seeder.
3. Peer continues until all chunks are successfully retrieved.
4. Peer notifies the tracker of completion.

5. Concurrent Downloads

1. Peer maintains separate chunk buffers for each file.
2. Peer initiates concurrent download tasks for each file.
3. Each task independently handles chunk requests and responses.
4. Peer completes all downloads and assembles files.

6. System structure

The P2P file-sharing system is defined in the `docker-compose.yml`

6.1. Services

Tracker

- Acts as the central coordinator for peers.
- Keeps track of available torrents, seeders, and leechers.
- Facilitates communication by providing metadata to peers.

- **Configuration**:
  - **Ports**: Exposed on `8888`.
  - **Command**: Runs the `tracker.py` script to manage torrents.
  - **Network**: Static IP (`172.20.0.2`) on the `p2p_network`.
  - **Volumes**: Shares the `src` directory for source code.

Seeder

- Hosts the initial copy of the file.
- Provides file chunks to requesting peers (leechers).

- **Configuration**:
  - **Ports**: Exposed on `8001`.
  - **Command**: Runs `client_handler.py`, specifying its address (`172.20.0.3`), tracker IP (`172.20.0.2`), and port (`8888`).

- **Volumes**:
  - Maps `./input` to `/app/files` for seeding files.
  - Maps `./output` to `/app/output` for logs or processed files.
- **Network**: Static IP (`172.20.0.3`) on the `p2p_network`.
- **Dependencies**: Waits for the `tracker` service to start.

Leecher1

- Requests chunks of files from the seeder, then seeds the downloaded file.
- Downloads and reassembles the file in its local `output` folder.

- **Configuration**:
  - **Ports**: Exposed on `8002`.
  - **Command**: Runs `client_handler.py`, specifying its address (`172.20.0.4`), tracker IP (`172.20.0.2`), and port (`8888`).
  - **Volumes**:
    - Maps `./output/leecher1` to `/app/output` for storing downloaded files.
  - **Network**: Static IP (`172.20.0.4`) on the `p2p_network`.
  - **Dependencies**: Depends on both `tracker` and `seeder`.

Leecher2

- Another peer that downloads file chunks, in this case it will download the chunks from both other peers.

- **Configuration**:
  - **Ports**: Exposed on `8003`.

  - **Command**: Runs `client_handler.py`, specifying its address (`172.20.0.5`), tracker IP (`172.20.0.2`), and port (`8888`).
  - **Volumes**:
    - Maps `./output/leecher2` to `/app/output` for storing downloaded files.
  - **Network**: Static IP (`172.20.0.5`) on the `p2p_network`.
  - **Dependencies**: Depends on both `tracker` and `seeder`.

6.2. Network

- **Name**: `p2p_network`
- **Type**: `bridge`
- **Subnet**: `172.20.0.0/16`

6.3. Input and Output Folders

- **Input Folder**:
  - Location: `./input`
  - Mounted on the seeder as `/app/files`.
  - Contains files to be shared with peers.
- **Output Folders**:
  - Locations:
    - `./output/seeder`: Files downloaded by seeder.
    - `./output/leecher1`: Files downloaded by `leecher1`.
    - `./output/leecher2`: Files downloaded by `leecher2`.
  - Each peer has its own output directory for clarity and isolation.

7. Running the system:

In order to run the system you'd need Python3.8+ and Docker installed.

The default setup is a tracker, one seeder and two leechers. (leecher1, leecher2)

```
docker-compose up --build
```

-	project	-	-
8888:8888	p2p_tracker	1bb69b68bf9d	project-tracker
8001:8001	p2p_seeder	06c9012d3912	project-seeder
8003:8003	p2p_leecher2	502a5ec6156e	project-leecher2
8002:8002	p2p_leecher1	19a395915454	project-leecher1

To Attach to peer's console:

```
docker attach <docker_name>
```

```
p2p file sharing client
------------------------------
[1] List available torrents
[2] Download a file
[3] Share a file
```

Uploading a file:

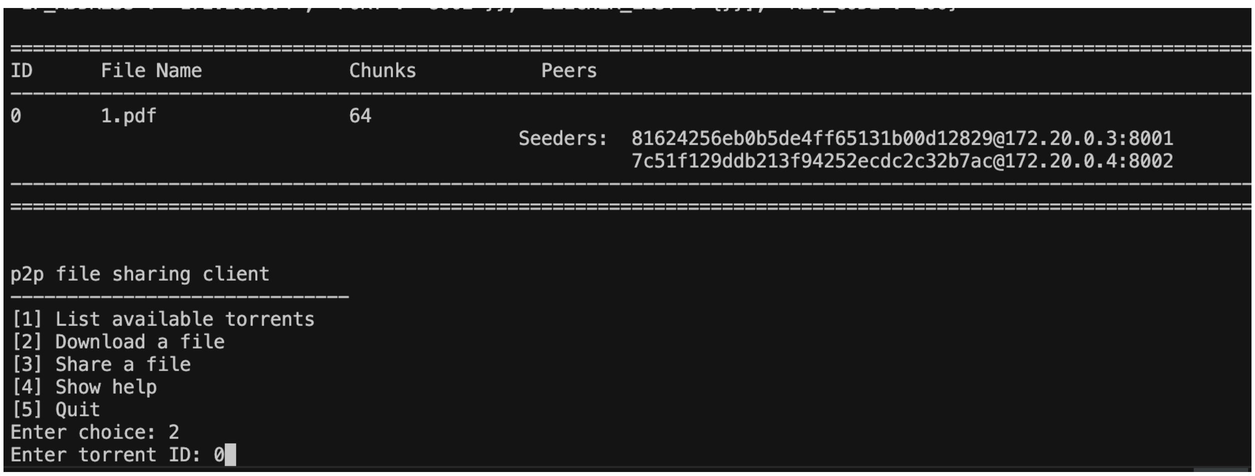
```
p2p file sharing client
------------------------------
[1] List available torrents
[2] Download a file
[3] Share a file
[4] Show help
[5] Quit
Enter choice: 3
Enter filename: input/1.pdf
2024-12-06 13:54:35,727 - INFO - uploading file as seeder input/1.pdf
2024-12-06 13:54:35,735 - DEBUG - sending message: {"OP_CODE": 140, "IP_ADDRESS": "172.20.0.3", "PORT": "8001", "PEER_ID": "81624256eb0b5de
f65131b00d12829", "FILE_NAME": "1.pdf", "NUM_OF_CHUNKS": 64}
2024-12-06 13:54:35,736 - INFO - receiving request from handler
2024-12-06 13:54:35,736 - DEBUG - Reading message
2024-12-06 13:54:35,740 - DEBUG - Received message: {'OP_CODE': 140, 'RET_CODE': 200, 'TORRENT_ID': 0}
2024-12-06 13:54:35,740 - INFO - Starting seeding server on ('172.20.0.3', 8001)
```

Listing available files:

```
Enter choice: 1
2024-12-06 13:55:28,431 - DEBUG - sending message: {"OP_CODE": 100, "IP_ADDRESS": "172.20.0.3", "PORT": "8001", "PEER_ID": "81624256eb0b5de4f
f65131b00d12829"}
2024-12-06 13:55:28,435 - INFO - receiving request from handler
2024-12-06 13:55:28,436 - DEBUG - Reading message
2024-12-06 13:55:28,446 - DEBUG - Received message: {'OP_CODE': 100, 'TORRENT_LIST': [{'TORRENT_ID': 0, 'FILE_NAME': '1.pdf', 'NUM_OF_CHUNKS'
: 64, 'SEEDER_LIST': {'81624256eb0b5de4ff65131b00d12829': {'IP_ADDRESS': '172.20.0.3', 'PORT': '8001'}}, 'LEECHER_LIST': {}}], 'RET_CODE': 20
0}

========================================================================================================================
ID      File Name              Chunks          Peers
------------------------------------------------------------------------------------------------------------------------
0       1.pdf                  64
                                             Seeders:  81624256eb0b5de4ff65131b00d12829@172.20.0.3:8001
------------------------------------------------------------------------------------------------------------------------
========================================================================================================================
```

Downloading a file:



```
=========================================================================================================
ID      File Name              Chunks           Peers
---------------------------------------------------------------------------------------------------------
0       1.pdf                  64
                                              Seeders:  81624256eb0b5de4ff65131b00d12829@172.20.0.3:8001
                                                        7c51f129ddb213f94252ecdc2c32b7ac@172.20.0.4:8002
---------------------------------------------------------------------------------------------------------
=========================================================================================================


p2p file sharing client
------------------------------
[1] List available torrents
[2] Download a file
[3] Share a file
[4] Show help
[5] Quit
Enter choice: 2
Enter torrent ID: 0
```

Running the tests:

```
cd src/tests/
PYTHONPATH=.. python -m unittest discover
```

8. Error handling and Retry mechanism:

The system has an error handling and retry mechanism, to ensure all chunks are downloaded.

Once a chunk is failed to download, the system tries to retry it, and if possible even from other available seeds.

Retry from one seed:

```
2024-12-06 14:20:31,197 - INFO - Retrying read...
2024-12-06 14:20:31,704 - INFO - Retrying read...
2024-12-06 14:20:32,208 - INFO - Retrying read...
2024-12-06 14:20:32,711 - DEBUG - result in retry 450
2024-12-06 14:20:32,711 - ERROR - Failed to download chunk 38 from peer 0
2024-12-06 14:20:32,711 - INFO - connecting to seeder at 172.20.0.3:8001
2024-12-06 14:20:32,713 - INFO - connected as leecher: 172.20.0.4:8002
2024-12-06 14:20:32,713 - DEBUG - sending message: {"OP_CODE": 195, "IP_ADDRESS": "172.20.0.4", "PORT": "8002", "CHUNK_INDX": 39}
2024-12-06 14:20:32,713 - DEBUG - Reading message
2024-12-06 14:20:32,716 - DEBUG - Received message: {'OP_CODE': 195, 'IP_ADDRESS': '172.20.0.3', 'PORT': '8001', 'CHUNK_DATA': 'tw/QQOhEJqcnn
yeJn9+c...', 'CHUNK_INDX': 39, 'RET_CODE': 200}
2024-12-06 14:20:32,716 - DEBUG - result in retry 200
2024-12-06 14:20:32,717 - INFO - Successfully downloaded chunk 39 from peer 0
2024-12-06 14:20:32,717 - INFO - connecting to seeder at 172.20.0.3:8001
2024-12-06 14:20:32,718 - INFO - connected as leecher: 172.20.0.4:8002
2024-12-06 14:20:32,718 - DEBUG - sending message: {"OP_CODE": 195, "IP_ADDRESS": "172.20.0.4", "PORT": "8002", "CHUNK_INDX": 40}
2024-12-06 14:20:32,718 - DEBUG - Reading message
2024-12-06 14:20:32,720 - DEBUG - Received message: {'OP_CODE': 195, 'IP_ADDRESS': '172.20.0.3', 'PORT': '8001', 'CHUNK_DATA': 'lPMeOAdi+eeAD
```

Retry from different peers:

```
2024-12-06 14:22:35,763 - INFO - Successfully downloaded chunk 61 from peer 1
2024-12-06 14:22:35,763 - INFO - connecting to seeder at 172.20.0.3:8001
2024-12-06 14:22:35,764 - INFO - connected as leecher: 172.20.0.5:8003
2024-12-06 14:22:35,764 - DEBUG - sending message: {"OP_CODE": 195, "IP_ADDRESS": "172.20.0.5", "PORT": "8003", "CHUNK_INDX": 62}
2024-12-06 14:22:35,764 - DEBUG - Reading message
2024-12-06 14:22:35,766 - DEBUG - Received message: {'OP_CODE': 195, 'IP_ADDRESS': '172.20.0.3', 'PORT': '8001', 'CHUNK_DATA': 'XqCaPJWOrsSgH
GX6XKS+...', 'CHUNK_INDX': 62, 'RET_CODE': 200}
2024-12-06 14:22:35,767 - DEBUG - result in retry 200
2024-12-06 14:22:35,767 - INFO - Successfully downloaded chunk 62 from peer 0
2024-12-06 14:22:35,767 - INFO - connecting to seeder at 172.20.0.4:8002
2024-12-06 14:22:35,768 - INFO - connected as leecher: 172.20.0.5:8003
2024-12-06 14:22:35,768 - DEBUG - sending message: {"OP_CODE": 195, "IP_ADDRESS": "172.20.0.5", "PORT": "8003", "CHUNK_INDX": 63}
2024-12-06 14:22:35,768 - DEBUG - Reading message
2024-12-06 14:22:35,769 - DEBUG - Received message: {'OP_CODE': 195, 'IP_ADDRESS': '172.20.0.4', 'PORT': '8002', 'CHUNK_DATA': 'uCSgaKFstMxhX
vRkyA3m...', 'CHUNK_INDX': 63, 'RET_CODE': 200}
2024-12-06 14:22:35,769 - DEBUG - result in retry 200
2024-12-06 14:22:35,769 - INFO - Successfully downloaded chunk 63 from peer 1
2024-12-06 14:22:35,769 - INFO - Retry attempt 1 for chunks: {56}
2024-12-06 14:22:36,775 - INFO - connecting to seeder at 172.20.0.3:8001
```

9. File Handling

In order to support the P2P communication, we split large files into chunks, ensure recieving all chunks, and their integrity, then build the file again.

9.1. File Splitting (file_handler.py -> split_file)

How It Works

- **Chunk Size**:
  - Defined by `CHUNK_SIZE` in the protocol (e.g., 16KB or 24KB).
  - Files are read in chunks of this size.
- **Splitting Logic**:
  - Files are processed in binary mode (`rb`).
  - Each chunk is written to a separate file (e.g., `file_chunk_0`, `file_chunk_1`, etc.).
  - The splitting process calculates and stores a hash for each chunk to ensure data integrity during transfers.

9.2. Chunk Hashing (file_handler.py -> calculate_hash, verify_chunk)

Hashing ensures data integrity by verifying that chunks are not corrupted during transfer.

How It Works

- **Hashing Algorithm**:
  - Uses `SHA-256` for strong cryptographic hashing.
  - Reads chunks in blocks (e.g., 4KB) to calculate the hash.
- **Verification**:
  - When a chunk is received, its hash is recalculated and compared to the expected hash.
  - If the hashes match, the chunk is valid; otherwise, it's marked as corrupted.

9.3. Encoding and Decoding

- **Encoding**:
  - Chunks are base64-encoded before transmission to avoid issues with binary data.
- **Decoding**:
  - Received chunks are decoded back into their original binary form before storage.

10. Conclusion and Future Work

10.1. Future work:

1. Enhanced Error Handling: Implement more robust mechanisms to handle network failures, corrupted data, and tracker unavailability to ensure seamless operation under adverse conditions.
2. Encryption Support: Introduce end-to-end encryption for data transfer to ensure secure communication between peers and protect sensitive information.
3. Better Peer Selection: Develop algorithms to optimize peer selection, prioritizing peers with faster connections and better availability for efficient file sharing.
4. Web Interface/ GUI: Create a user-friendly web-based interface for easier management of torrents, peers, and file transfers, enhancing the system's usability.
5. NAT Traversal: Incorporate techniques to enable peers behind Network Address Translation (NAT) devices to connect directly without additional configuration.

6. Efficiency: This project is written in python for convenience, should consider more robust language that support efficient I/O and network communications.

10.2. Conclusion

The P2P File Sharing Project lays the groundwork for a scalable, efficient, and reliable system that leverages modern networking techniques. By addressing the proposed future enhancements, the system can achieve greater robustness, security, and usability. These improvements will ensure it remains competitive and capable of meeting evolving user demands in a decentralized sharing ecosystem.