VIETNAM NATIONAL UNIVERSITY, HO CHI MINH CITY
UNIVERSITY OF TECHNOLOGY
FACULTY OF COMPUTER SCIENCE AND ENGINEERING

Computer Networks (CO3093)

Assignment 1

Simple Torrent-like Application (STA)

Instructor: Nguyễn Phương Duy

Students:	Nguyễn Trung Tín	2213500
	Trần Quốc Toàn	2213540
	Nguyễn Thị Bảo Trâm	2213574
	Trương Anh Tuấn	2213810

HO CHI MINH CITY, DECEMBER 2024

Mục lục

1 Requirement elicitation

1.1 Domain Context

Before the widespread adoption of the internet, sharing digital content relied on physical storage devices such as floppy disks and USB drives. For long distances, these storage devices were often transported through postal services. These methods were constrained by delays, physical risks to the storage medium, and limited scalability.

With the advent of the internet, Peer-to-Peer (P2P) systems introduced a more efficient and decentralized approach to file sharing. P2P protocols, such as those used by BitTorrent, optimized bandwidth utilization and eliminated the need for centralized servers by distributing file chunks across multiple peers. These systems allowed users to upload and download content directly from one another, providing flexibility and reliability.

This project, Simple Torrent-like Application (STA), seeks to implement a basic P2P system inspired by BitTorrent. The application will focus on key functionalities such as file chunking, peer discovery, and distributed file sharing to explore the foundational principles of decentralized systems.

1.2 Object

The goal of this project is to create an application that enables users to share files through a peer-to-peer (P2P) network. The application will include the following features:

- Support multidimensional data transfer (MDDT), which allows users to download multiple files at the same time from different sources.
- Build a system with two main types of participants: a tracker, which acts as a server to keep track of nodes, and nodes, which are the clients that share files.
- Implement functionality to track nodes and manage the allocation of file pieces among them using the tracker protocol.
- Enable users to download files from other nodes as well as upload files for sharing with others in the network.

This system aims to make file sharing faster, more efficient, and user-friendly by leveraging the capabilities of P2P networking.

1.3 Functional Requirements

The application includes the two types of hosts: tracker and node.

1. **Node**

   (a) **Client**

   - When require a file that does not belong to its repository, a request is sent to the tracker to get the peers and files information.
   - Inform the tracker as to what files are contained in its local repository but does not actually transmit file data to the tracker through tracker protocol.
   - Connect to an active peer at a specified IP and port.
   - Select files to download or upload.
   - Upload files to multiple peers simultaneously.
   - Download files from peers into a specified local directory (don't send a request for a piece the peer does not have, and don't send a request for a piece the node already have.).
   - Display download/upload progress.
   - Manage downloaded files (e.g., delete them).
   - Support downloading multiple files from multiple source nodes at once, simultaneously.

   (b) **Server**

   - Maintain connections with active peers' clients
   - Respond to client download requests for pieces.
   - Enable multi-client file sharing using multithreading to handle concurrent transfers.

   (c) **User Interface**

   - Satisfy the minimum requirements of downloading and seeding a single torrent to multiple peers.
   - Provide ability to see all the upload/download statistics.

2. **Tracker**

   - Track file metadata, including file pieces and list of peers.
   - Maintain the active/inactive status of peers.

- Manage the availability of shared files.
- **Tracker request parameters**: should include the compacted flag(s) or metainfo in your request, a bare minimum requirement is a magnet text of your torrent that is able to parse (to locate to a Metainfo File placed on the server).
- **Tracker response**: respond with "text/plain" document consisting of a dictionary with the following keys:
  - *Failure reason*: If present, then no other keys may be present. The value is a human-readable error message as to why the request failed (string).
  - *Warning message*: (new, optional) Similar to failure reason, but the response still gets processed normally. The warning message is shown just like an error.
  - *Tracker id*: A string that the client should send back on its next announcements. If absent and a previous announcement sent a tracker id, do not discard the old value; keep using it.
  - *Peers*: (dictionary model) the value is a list of dictionaries, each with the following keys:
    - peer id: peer's self-selected ID, as described above for the tracker request (string)
    - ip: peer's IP address either IPv6 (hexed) or IPv4 (dotted quad) or DNS name (string)
    - port: peer's port number (integer)

1.4 Non-functional Requirements

1. **Availability**

ID	Description
NF-AVAI1	Ensure system availability of 99.9% uptime, with minimal scheduled downtime for maintenance
NF-AVAI2	Enable seamless resumption of interrupted downloads after application restarts

2. **Performance**
3. **Usability**
4. **Reliability**

ID	Description
NF-PERF1	Support a minimum of 10 concurrent download processes and up to 1,000 peer connections without significant performance degradation
NF-PERF2	Achieve a download speed of at least 80% of the user's available bandwidth under optimal conditions

ID	Description
NF-USAB1	Provide an intuitive and visually clear interface that displays progress and file size
NF-USAB2	Design a user-friendly interface for general users with accessible controls

ID	Description
NF-RELI1	Automatically reconnect to peers in the event of a disconnection
NF-RELI2	Implement robust data recovery mechanisms to prevent data loss during unexpected shutdowns

5. **Efficiency**
6. **Security**
7. **Compatibility**
8. **Scalabilty**
9. **Maintainability**
10. **Compliance**

ID	Description
NF-EFFI1	Optimize resource usage to minimize CPU and memory consumption, particularly on devices with limited hardware resources

ID	Description
NF-SECU1	Support end-to-end encryption for all data transfers to ensure privacy and security

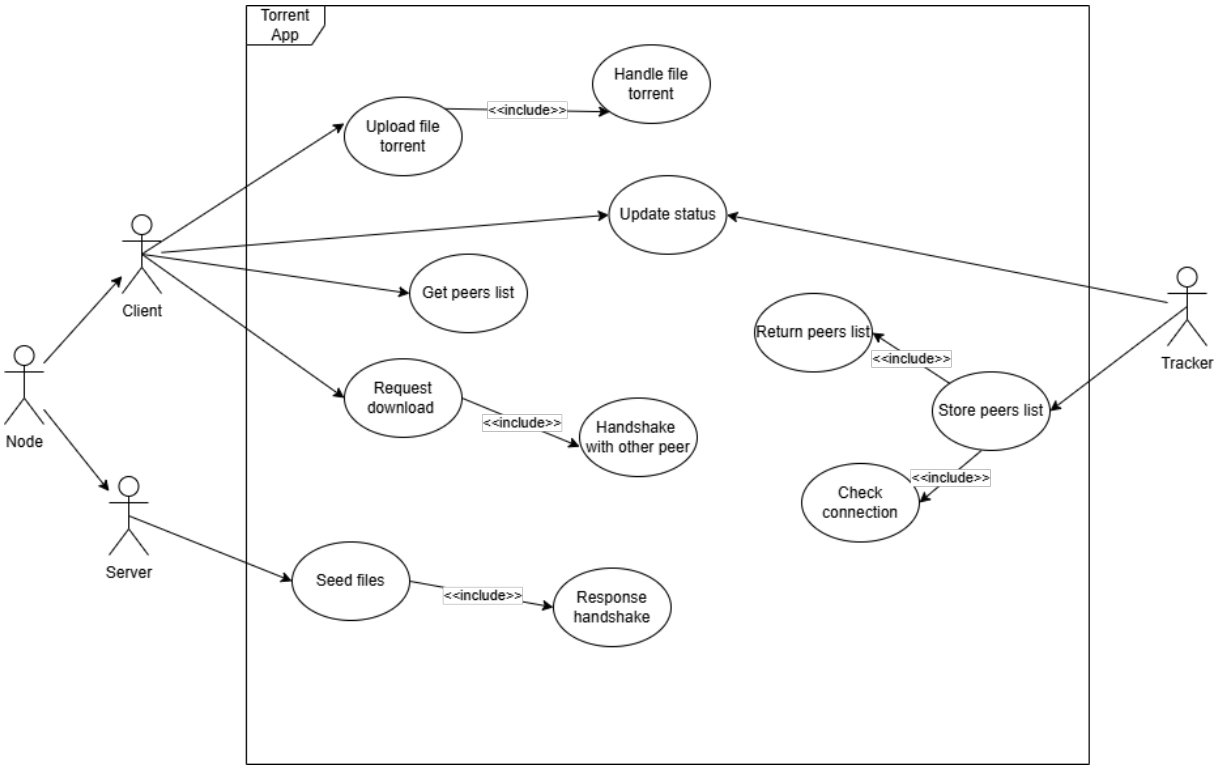
ID	Description
NF-COMP1	Ensure support for standard file-sharing formats such as .torrent and magnet links
NF-COMP2	Maintain compatibility with major operating systems, including Windows, macOS, and Linux

ID	Description
NF-SCAL1	Design the system to handle increases in the number of users and peer connections without requiring substantial architectural changes

ID	Description
NF-MAIN1	Provide comprehensive and clear logging mechanisms to facilitate debugging and issue resolution

ID	Description
NF-COMP1	Adhere to BitTorrent protocol standards and comply with applicable regional copyright regulations

1.5 Use-case Diagram



2 System Modeling

2.1 System Architecture

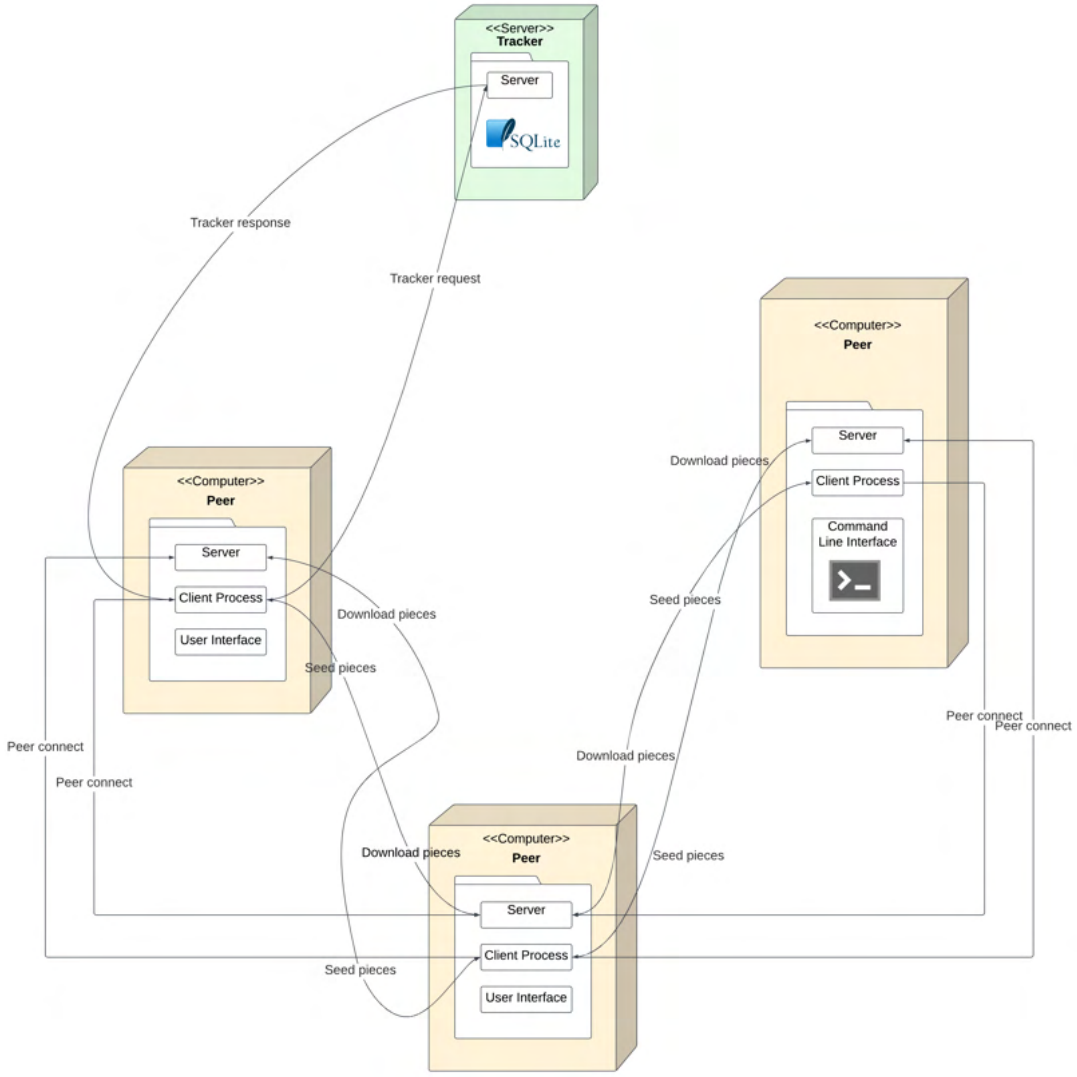
2.2 System functionalities and Protocols

2.2.1 Peer Discovery and Management

- **Functionality**: Use a centralized server, called a tracker, to maintain a record of all peers and the files they share. The tracker facilitates peer discovery and identifies available files for download.
- **Protocol**: Communication with Tracker via **HTTP**: Peers connect to the tracker using announce requests to retrieve a list of available peers.

2.2.2 File Sharing

- **Functionality**: Users can share files by uploading them to their local repository. The file metadata is recorded in the tracker, making it accessible for other peers.
- **Protocol**: Communication with Tracker via **HTTP**.



2.2.3 File Downloading

- **Functionality**: Peers request the tracker for a list of available peers holding the desired file and download the file directly from them.
- **Protocol**: Communication with Tracker via **HTTP**: Safeguard file-related data through encrypted transmission.

2.2.4 Network Communication

- **Functionality**:
  - Direct peer-to-peer connections for file transfers.
  - Secure communication with the tracker for metadata and peer information.

- **Protocol**: **TCP/IP** for direct peer connections: Ensure reliable and sequential delivery of data packets.

2.2.5 Activity Monitoring

- **Functionality**: Monitor upload/download status and provide real-time statistics.
- **Protocol**: **HTTP** for periodic updates:
  - Peers send periodic status updates (e.g., upload/download progress) to the tracker.
  - The tracker stores and responds with aggregated data for monitoring purposes.

2.3 Implement functions and protocols

- **Torrent Files Information**: The torrent file contains metadata, including information such as tracker URL, piece hashes, piece length, and file name.

2.3.1 Functional Description

- **Node (Client)**:
  - **AnnounceToTracker()**: Notifies a tracker about a file a peer wants to download/upload.
  - **connect_to_tracker()**: Establishes a connection to a tracker to send metadata about files being shared.
  - **tracker_request()**: Sends requests to the tracker to update information about the peer's progress (e.g., pieces downloaded).
  - **disconnect_from_tracker()**: Closes the connection to the tracker when it is no longer needed.
  - **handle_peer_connections()**: Manages incoming and outgoing connections with other peers in the network.
  - **request_piece()**: Sends a request to a peer to obtain a specific piece of the file.
  - **send_piece()**: Responds to a peer's request by sending the requested piece of the file.
  - **update_peer_status()**: Updates the status of connected peers, such as their availability and download/upload speeds.
  - **piece_verification()**: Verifies the integrity of a downloaded piece using its hash value.
  - **manage_download_queue()**: Handles the order in which file pieces are downloaded.
- **Node (Server)**:
  - **TorrentFile()**: Represents metadata for a torrent file, including tracker URL, info hash, piece hashes, piece length, file length, and file name.
  - **FileWorker()**: Manages file pieces, computes SHA-1 hashes for each piece, and provides access to the pieces.

- **get_host_default_interface_ip()**: Retrieves the default IP address of the host machine.
- **generate_peer_id_with_ip()**: Generates a unique peer ID based on the client prefix and the host IP address.
- **handle_connection()**: Manages incoming client connections, processes handshake and piece request messages.
- **new_file_worker()**: Creates a new instance of FileWorker for the specified file path.
- **handle_handshake()**: Processes handshake messages from peers, validates the info hash, and associates it with a file worker.
- **handle_piece_request()**: Validates and serves a requested piece of the file, including its size header, to the client.
- **getTorrentFiles()**: Retrieves the list of .torrent files available in the torrent_files directory.
- **start_server()**: Starts the server, listens for client connections, and spawns threads to handle each connection.

- **Tracker**:

Tracker_Database	
PK	Info_hash, peer_id
	ip: Text
	port: Integer
	downloaded: Integer
	left: Integer
	last_seen: Real

- **init_db()**: Initializes the SQLite database by creating the peers table if it doesn't exist.
- **upsert_peer(info_hash, peer_id, ip, port, downloaded, left)**: Adds or updates a peer's information in the database, including the last time it communicated with the tracker.

- **get_peer_list(info_hash)**: Retrieves the list of peers associated with a specific torrent based on the info_hash.
- **delete_inactive_peers(inactivity_threshold=3600)**: Removes peers that haven't communicated with the tracker within the specified threshold (default is 1 hour).
- **announce()**: Handles GET requests for the /announce endpoint, managing events like starting, completing, or stopping a torrent session, and returning a list of active peers.
- **periodic_cleanup()**: A thread that periodically removes inactive peers from the database every 10 minutes.

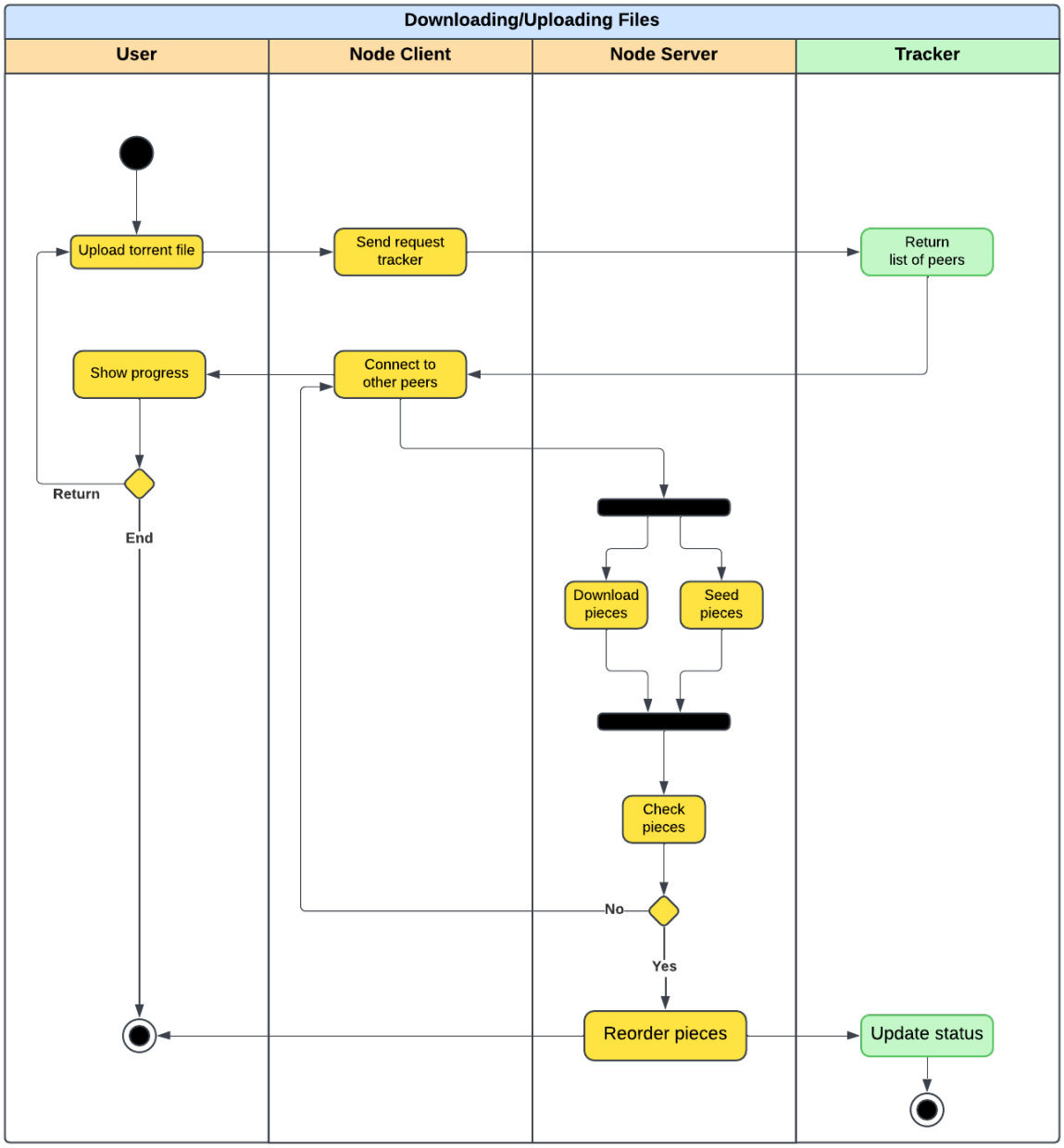
2.3.2 Protocols

- **Protocol Between Node and Tracker**: The features involving communication between the node and tracker will use the HTTP connection protocol. The Tracker is built using the HTTP protocol, where it handles client requests through HTTP GET requests. In response, it provides a list of peers that have the file or specific pieces of the file the client is searching for. The peer and file data is stored in an SQLite database.
  - **Seeding To Tracker**: The client starts by calling the Seed() function with the parameters peer_id, peer_ip, and the torrent_file wish to seed. The torrent file will be analyzed to extract information such as URL (announce), the torrent's unique identifier (info_hash), and the total length of the file being shared (total_length). The tracker_request() sends an HTTP GET request to the tracker with parameters. This allows the tracker to process the client's request, update its records, and send a response.
  - **Get Peer List**: In the Download() function when sending a tracker request. The function extracts the tracker_url from the torrent file and sends an HTTP GET request with parameters like info_hash and peer_id. The tracker responds with a list of peers, enabling the client to connect and download the file.
  - **Disconnect To Tracker**: The disconnect_to_tracker() function iterates through the connected_tracker_addresses list, and for each tracker, it sends an HTTP request via tracker_request. This request includes the tracker's URL (tracker_url),

the peer_id, the peer_ip, and the event "stopped". The HTTP request notifies the tracker that the client has stopped participating in the torrent.

- **Protocol Between Node (Server)**: After retrieving the peer information from the tracker, the peers begin exchanging data with each other using the TCP connection protocol. Each peer utilizes the TCP protocol to establish connections with other peers. They send requests to download specific file pieces while also offering to share their own file pieces with others upon request. Data transfer between peers happens through multiple concurrent connections, enhancing download speeds. Files are split into smaller pieces, with details about these pieces stored in a database. A peer only needs to download a subset of pieces, which it can then share, ensuring efficient resource distribution.
  - **Downloading File**: After retrieving the list of peers from the tracker, the node begins searching for active peers and starts downloading from them. The TCP connection will be established through a handshake, the client and peer exchange file pieces over TCP, ensuring reliable data transfer. If any packet is lost or corrupted, TCP guarantees retransmission. After the file is downloaded, the connection is gracefully closed, ensuring that all data is received correctly and in order.

2.4 Activity Diagram



This diagram describes a workflow where the user initiates a file sharing session, the peer client connects to other peers to download file pieces, the peer server facilitates file seeding and piece verification, and the tracker coordinates the peer network by managing and updating peer lists.

The user upload file torrent to retrieve the file's metadata. Upon starting a download, the peer client sends a request to the tracker for a list of peers that hold the file pieces.

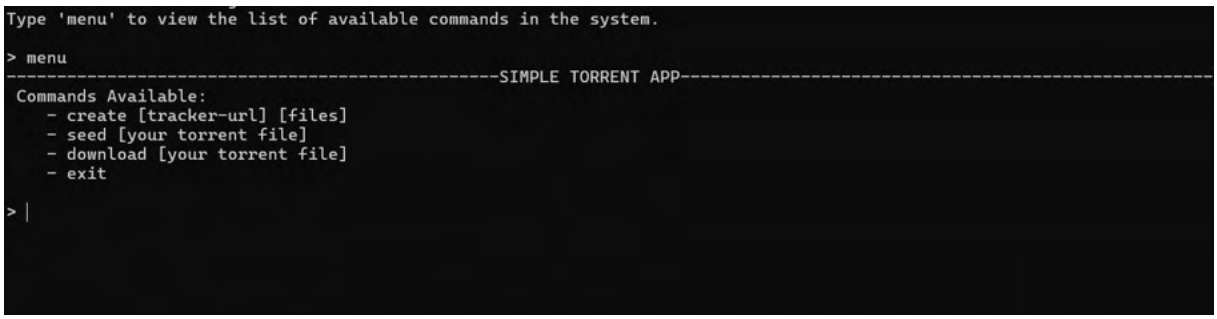
Using the list from the tracker, the peer client connects to other peers in the network to begin the download. If any issues occur during the download process, such as a missing piece or failed connection, the peer client alerts the user. As the upload or download process proceeds, progress is displayed to the user. Once pieces are downloaded, the peer server seeds them to other clients, maintaining the file's availability. The server verifies the integrity of downloaded pieces to ensure they match the original file. If pieces are out of order or missing, the server reorders or requests missing parts to complete the file.

The tracker receives requests from peer clients and returns a list of available peers with the requested file pieces. Also, the tracker updates the availability status of pieces and peer connections to help maintain efficient file sharing.

3 Demonstration

3.1 Command Line Interface (CLI)

- In the Command Line Interface of the peers, there will be commands provided by the system.



- **create** [**tracker-url**] [**files**]: The command is used to generate a new torrent file associated with the specified tracker URL and the list of files to be shared.
- **seed** [**your torrent file**]: command is used to start sharing the specified torrent file, allowing the peer to act as a seeder and provide file pieces to other peers in the network.
- **download** [**your torrent file**]: The download [your torrent file] command is used to initiate downloading the file associated with the specified torrent file. The peer connects to other peers in the network to fetch the file pieces and reassemble the complete file.
- **exit**: Used to terminate the program, disconnect from the tracker, and shut down the peer's operations gracefully.

3.2 File Transfer Within A Subnet

A subnet will be set up consisting of 4 computers: one as the tracker, two performing the seeding function, and one performing the download function.

3.2.1 Run Tracker

- The Computer Hosting The Tracker Using Port 4000.

```
* Running on all addresses (0.0.0.0)
* Running on http://127.0.0.1:4000
* Running on http://192.168.88.114:4000
```

3.2.2 Run Peer

- The Three Computers Using As The Peer Uses Port 8080.

```
Thread server listening on: 192.168.88.114:8080
```

```
Thread server listening on: 192.168.88.178:8080
```

```
[Thread server listening on: 192.168.88.102:8080
```

- If there is no torrent file, we can execute the create command with the syntax above. The torrent file will be named according to the name of the file added if only one file is included. If multiple files are added, the name will be generated randomly. The file will be saved in the folder name "torrent_files"

```
> create 192.168.88.114 ass1.pdf
Successfully creating ass1.torrent

> create 192.168.88.114 ass1.pdf report.pdf
Successfully creating cda26562ce77c6f83870b7e24235d92a6030c597.torrent
```

- **Seeding**: Two computers share the same file.

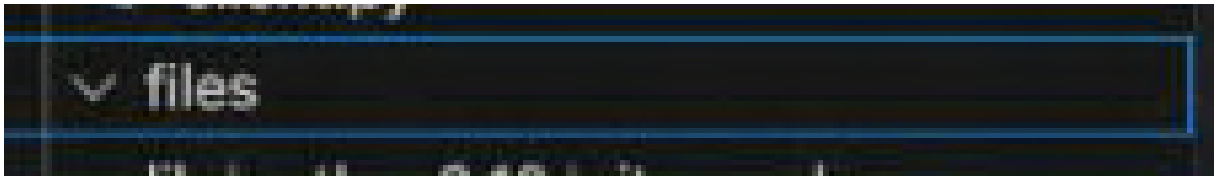
```
> seed ass1.torrent
Tracker response:
info_hash=1cfa5de9ecd67c87d1cc2ff2464aef0e3051d2e5
peers=192.168.88.114
```

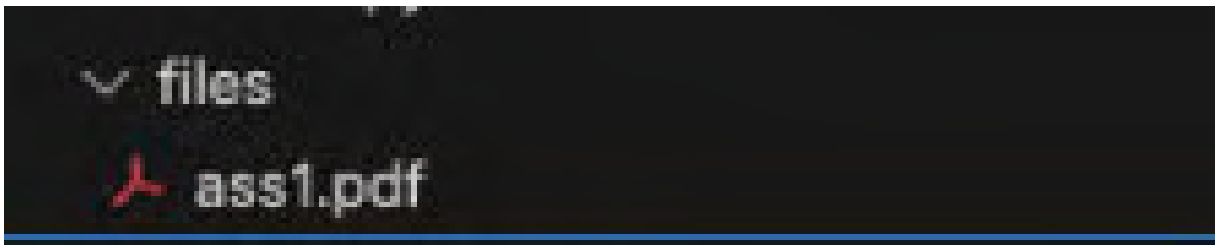
```
> seed ass1.torrent
Tracker response:
info_hash=1cfa5de9ecd67c87d1cc2ff2464aef0e3051d2e5
peers=192.168.88.178,192.168.88.114
```

- **Downloading**: The computer performs downloading function.

```
> download ass1.torrent
[{'announce': 'http://192.168.88.114:4000', 'info_hash': 'e
': 232142, 'name': 'ass1.pdf'}]
Tracker response:
info_hash=1cfa5de9ecd67c87d1cc2ff2464aef0e3051d2e5
peers=192.168.88.178,192.168.88.102,192.168.88.114
['192.168.88.178', '192.168.88.102', '192.168.88.114']
Handshake successful!
Thread 6212120576: Added 192.168.88.114 to active peers
```

- The file will be saved in the folder named "files."





- **On The Peer Seed Side**: it will receive requests from the peer downloader.

```
> Connection from ('192.168.88.102', 63159)
Received message: HANDSHAKE:1cfa5de9ecd67c87d1cc2ff2464aef0e3051d2e5
ass1.pdf
[b'\xbbbV\xad\x116\xde]m\xccG{\xc5\xcdL\xe2kC\xfeM']
{'e776e31639cddcc90a567c9b698fd5fe87169c7c': <server.server.FileWorker o
Connection from ('192.168.88.102', 63160)
Received message: Requesting:e776e31639cddcc90a567c9b698fd5fe87169c7c:0
```

- In this log, a peer connects to the server, sending a handshake with the file identifier and the file name. After the initial handshake, the peer requests a specific piece of the file identified by its info hash. The server responds by handling the file-related request and ensuring that the file is transferred securely, likely using hash validation.

3.2.3 Multidimensional Data Transfer

- If a peer wants to download multiple files, it can retrieve them from various sources using the multidimensional data transfer (MDDT) mechanism.
- Peer 1: Share "ass1.pdf"

```
> create 192.168.88.179 ass1.pdf
Successfully creating ass1.torrent

> seeding ass1.torrent
Tracker response:
info_hash=1cfa5de9ecd67c87d1cc2ff2464aef0e3051d2e5
peers=192.168.88.179
```

- Peer 2: Share "report.pdf"

```
> seeding report.torrent
Tracker response:
info_hash=f72d79db09dc4812bb831941e6b49cdb66bd1b27
peers=192.168.88.102
```

- Peer 3: Download "ass1.pdf" and "report.pdf"

```
'name': 'report.pdf'}]
Tracker response:
info_hash=cab7f783ff5ae5cab6ee1b682050a0817f4d7399
peers=192.168.88.102,192.168.88.114
['192.168.88.102', '192.168.88.114']
Handshake successful!
Thread 21568: Added 192.168.88.102 to active peers
Active peers after handshake: ['192.168.88.102']
['192.168.88.102']
Downloading file: ass1.pdf
Downloading piece 0 from peer 192.168.88.102
size_header:b'\x00\x00\x00\x00\x00\x03\x8a\xce'
Successfully downloaded piece 0 of ass1.pdf
Download complete for file: ass1.pdf
Tracker response:
info_hash=cab7f783ff5ae5cab6ee1b682050a0817f4d7399
peers=192.168.88.102,192.168.88.114
Downloading file: report.pdf
Downloading piece 0 from peer 192.168.88.102
Downloading piece 1 from peer 192.168.88.102
Downloading piece 2 from peer 192.168.88.102
Downloading piece 3 from peer 192.168.88.102
Downloading piece 4 from peer 192.168.88.102
Downloading piece 5 from peer 192.168.88.102
Downloading piece 6 from peer 192.168.88.102
size_header:b'\x00\x00\x00\x00\x00\x04\x00\x00'
size_header:b'\x00\x00\x00\x00\x00\x04\x00\x00'
size_header:b'\x00\x00\x00\x00\x00\x04\x00\x00'
size_header:b'\x00\x00\x00\x00\x00\x04\x00\x00'
size_header:b'\x00\x00\x00\x00\x00\x04\x00\x00'
size_header:b'\x00\x00\x00\x00\x00\x04\x00\x00'
size_header:b'\x00\x00\x00\x00\x00\x04\x00\x00'
Downloading piece 7 from peer 192.168.88.102
Downloading piece 8 from peer 192.168.88.102
Downloading piece 9 from peer 192.168.88.102
Downloading piece 10 from peer 192.168.88.102
Downloading piece 11 from peer 192.168.88.102
Downloading piece 12 from peer 192.168.88.102
Downloading piece 13 from peer 192.168.88.102
size_header:b'\x00\x00\x00\x00\x00\x04\x00\x00'
size_header:b'\x00\x00\x00\x00\x00\x04\x00\x00'
```

```
size_header:b'\x00\x00\x00\x00\x00\x04\x00\x00'
Downloading piece 7 from peer 192.168.88.102
Downloading piece 8 from peer 192.168.88.102
Downloading piece 9 from peer 192.168.88.102
Downloading piece 10 from peer 192.168.88.102
Downloading piece 11 from peer 192.168.88.102
Downloading piece 12 from peer 192.168.88.102
Downloading piece 13 from peer 192.168.88.102
size_header:b'\x00\x00\x00\x00\x00\x04\x00\x00'
size_header:b'\x00\x00\x00\x00\x00\x04\x00\x00'
size_header:b'\x00\x00\x00\x00\x00\x04\x00\x00'
size_header:b'\x00\x00\x00\x00\x00\x04\x00\x00'
size_header:b'\x00\x00\x00\x00\x00\x04\x00\x00'
size_header:b'\x00\x00\x00\x00\x00\x04\x00\x00'
size_header:b'\x00\x00\x00\x00\x00\x04\x00\x00'
Downloading piece 14 from peer 192.168.88.102
Downloading piece 15 from peer 192.168.88.102
size_header:b'\x00\x00\x00\x00\x00\x00d\xbf'
size_header:b'\x00\x00\x00\x00\x00\x04\x00\x00'
Successfully downloaded piece 1 of report.pdf
Successfully downloaded piece 4 of report.pdf
Successfully downloaded piece 2 of report.pdf
Successfully downloaded piece 0 of report.pdf
Successfully downloaded piece 5 of report.pdf
Successfully downloaded piece 3 of report.pdf
Successfully downloaded piece 6 of report.pdf
Successfully downloaded piece 9 of report.pdf
Successfully downloaded piece 8 of report.pdf
Successfully downloaded piece 10 of report.pdf
Successfully downloaded piece 11 of report.pdf
Successfully downloaded piece 7 of report.pdf
Successfully downloaded piece 12 of report.pdf
Successfully downloaded piece 13 of report.pdf
Successfully downloaded piece 15 of report.pdf
Successfully downloaded piece 14 of report.pdf
Download complete for file: report.pdf
Tracker response:
info_hash=cab7f783ff5ae5cab6ee1b682050a0817f4d7399
peers=192.168.88.102,192.168.88.114
```