VIETNAM NATIONAL UNIVERSITY, HO CHI MINH CITY

UNIVERSITY OF TECHNOLOGY

FACULTY OF COMPUTER SCIENCE AND ENGINEERING

COURSE: COMPUTER NETWORKS

SEMESTER 1 – ACADEMIC YEAR 2024-2025

REPORT - ASSIGNMENT 1

DEVELOP A NETWORK APPLICATION

Advisor: Diệp Thanh Đăng

Students: Huỳnh Gia Hưng – 2252274

Mạc Hồ Do Khang – 2252297

Nguyễn Đức Hạnh Nhi – 2252578

HO CHI MINH CITY, DECEMBER 2024

TABLE OF CONTENTS

1. Member list and Workload

No.	Fullname	Student ID	Workload	Percentage
1	Huỳnh Gia Hưng	2252274	- Client and Server Implementation - Write report	100%
2	Mạc Hồ Do Khang	2252297	- File and Piece Implementation - Write report	100%
3	Nguyễn Đức Hạnh Nhi	2252578	- Client and Server Implementation - Write report	100%

2. Overview

The purpose of this project is to design and implement a Simple Torrent-like Application (STA), modeled as a streamlined version of BitTorrent, using the TCP/IP protocol stack. The project aims to fulfill key requirements such as multi-direction data transfer (MDDT) and peer-to-peer (P2P) file sharing, focusing on efficient data dissemination among decentralized peers.

The STA operates using two core components: a centralized tracker and a network of nodes. The tracker maintains metadata about file availability and provides information to nodes seeking to download files. Nodes interact with the tracker and each other to achieve efficient MDDT by enabling simultaneous file transfers from multiple sources.

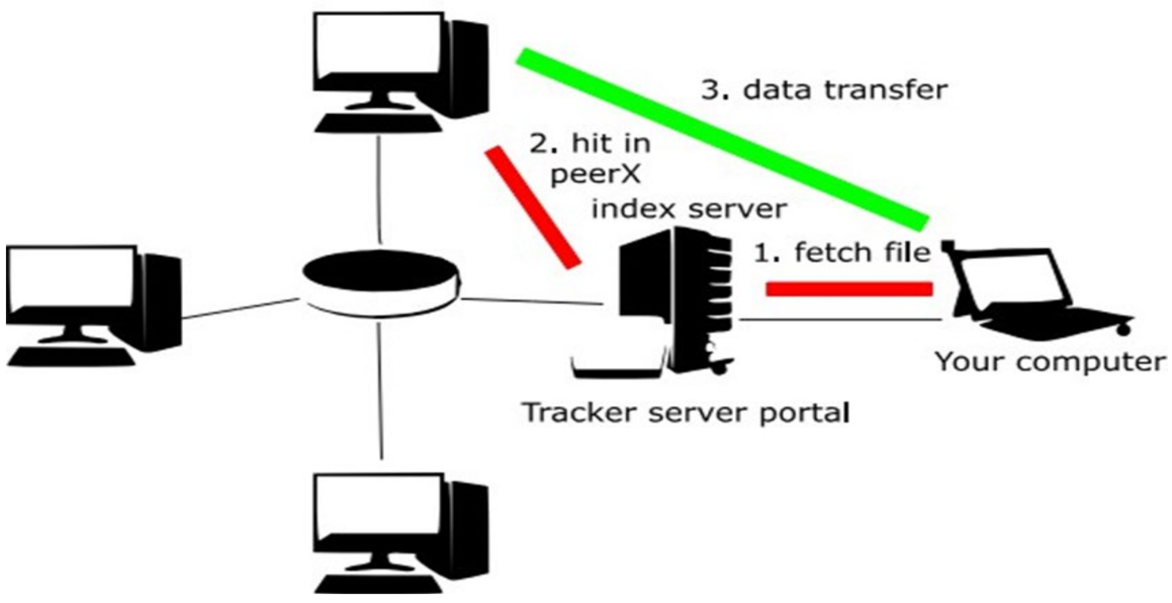


Figure 1. Illustration of file-sharing system

Key features of the STA include:

• **Tracker**: A centralized entity managing metadata and facilitating peer discovery without direct involvement in data transfer.

• **Nodes**: Decentralized peers capable of both downloading from and uploading to other nodes, contributing to the P2P network's scalability and redundancy.

• **MDDT**: A multithreaded implementation enabling concurrent data downloads from multiple sources to maximize transfer efficiency.

3. System Architecture

3.1. Main Components:

- **Tracker**:
    - Acts as the centralized metadata server.
    - Maintains information on available files and their respective chunks across the network.
    - Facilitates peer discovery by providing a list of active peers holding the requested file chunks.
    - Handles registration, updates, and status tracking of peers.
- **Peers**:
    - Function as nodes capable of both uploading and downloading files.
    - Inform the tracker about their available file chunks during registration.
    - Request chunks of files from the tracker or other peers.
    - Support multi-direction data transfer (MDDT) by simultaneously communicating with multiple peers.

3.2 Class Diagram

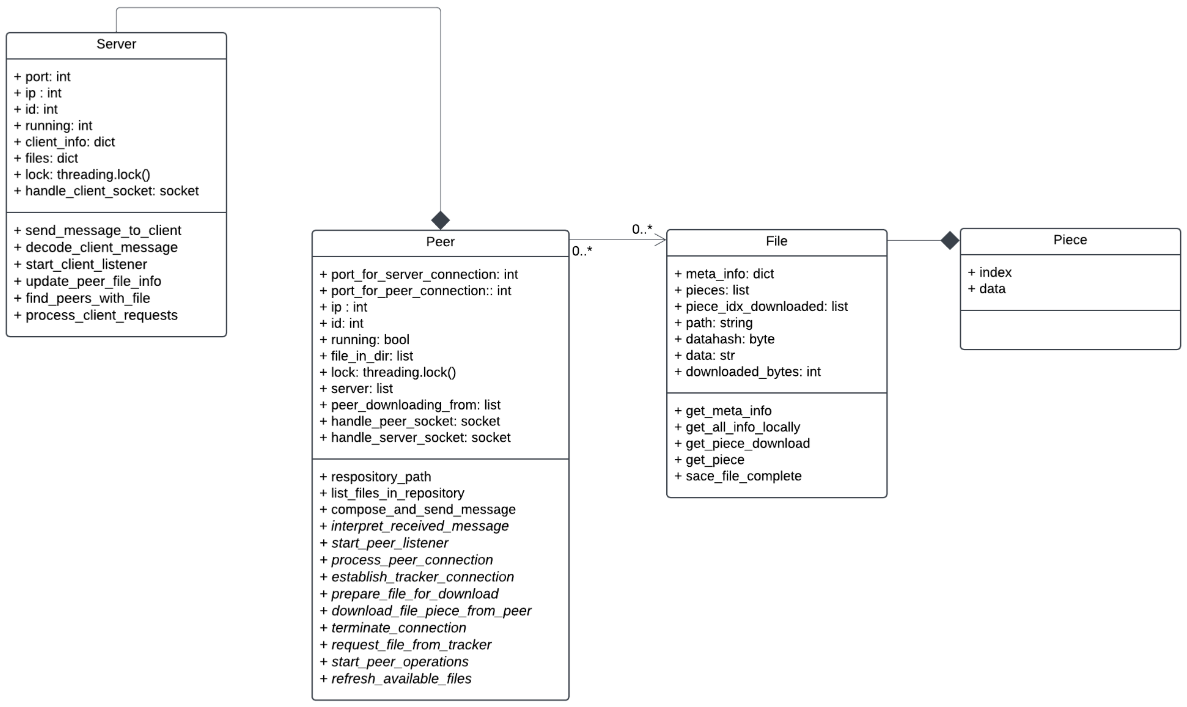


Figure 2. UML class diagram

The Server class has attributes to store a list of peers along with all the information about each peer's sharing files. The main attributes of the Server class are:

- client_info: This includes all the information about each client connecting to the server and information about files that each client is ready to share.

- files: This includes all the files that are available for downloading from other clients. The Server class also has methods to handle client connections, send messages, and process client requests.

The Peer class has attributes for the host, ports, and file list. It has methods to connect to the main server, handle sending requests to the server, sending messages to other clients, receiving requests, or uploading files for other clients. Each client is also capable of reading files from its own repository. Important features of the Peer class include:

- file_in_dir: This keeps all the information about files in its repository, which is used for submitting to the server and downloading pieces from others.

The Peer class also has methods to manage connections, handle file downloads, and refresh available files.

The Piece class has attributes to store its data of the file and index in the file chunks of pieces. The attributes of a Piece are:

- index: The index of the piece.

- data: The data contained in the piece.

The File class has attributes to store and manage the pieces of a file. This class also helps to read and write files from/to the peer's repository. The most vital aspects of the File class are:

- piece_idx_downloaded: This helps manage the status of the files by keeping track of downloaded pieces.

- piece_idx_not_downloaded: This helps manage the status of the files by keeping track of pieces not yet downloaded.

The File class also has methods to retrieve metadata, manage file pieces, and save the complete file.

3.3 Data Flow Explanation

Registration Phase:

- Peers connect to the tracker and register their file metadata, including available chunks.
- The tracker updates its database with peer information and the corresponding file chunk details.

File Request and Sharing Phase:

- A peer requests a specific file or chunk from the tracker.
- The tracker responds with a list of peers possessing the required chunk.

Chunk Downloading Phase:

- The requesting peer connects to one or more peers holding the desired chunks.
- Data is downloaded in parallel from multiple peers (MDDT), with acknowledgments sent to ensure completeness.

Upon completion, the peer notifies the tracker of its updated chunk availability for future sharing.

4. Components and Functions

4.1 Class Peer:

Represents a peer node in the torrent-like system. Each node can share files with other peers, download files from multiple peers simultaneously and communicate with the tracker and other peers.

Below are the methods implemented for class Peer:

- *__init__:* Initializes the peer node with IP Address and Ports, file repository, sockets and threads, peer state.
- *repository_path*: Returns the directory path where the peer’s files are stored. Creates the directory if it does not exist.
- *list_files_in_repository*: Scans the peer's repository directory and remove empty files, loads the metadata of existing files into memory for sharing or downloading and then returns a list of File objects representing the peer's local files.
- *compose_and_send_message*: Constructs and sends a structured message to a target socket. The message type determines its content:
  - **Handshake:** Shares peer ID and file list with another peer or the tracker.
  - **Request:** Requests specific file pieces from other peers.
  - **Response:** Sends file pieces or updates in response to a request.

- *interpret_received_message*: Decodes and parses messages received from peers or the tracker. Extracts key information such as message type, file details, and data. Handles different message formats based on the type.
- *start_peer_listener*: Listens for incoming connections from other peers and spawns threads to handle each connection. Uses a timeout to avoid indefinite blocking when no peers are connecting.
- *process_peer_connection*: Manages communication with a specific peer: Responds to file piece requests from the peer by checking the repository and sending the requested data and closes the connection if a CLOSE message is received.
- *establish_tracker_connection:* Connects to the tracker server to:
    - **Handshake:** Shares the peer's file list and ID with the tracker.
    - **Receive Peer List:** Requests and receives a list of peers having specific files.
    - **Update Tracker:** Sends updates to the tracker about changes in the peer's file repository.
- *prepare_file_for_download:* Prepares the file structure for downloading by creating an empty file in the repository and initializing metadata for the file, including total pieces and downloaded pieces.
- *download_file_piece_from_peer:* Downloads specific file pieces from a peer. The process includes: Requesting a piece by index, then cvrifying the received data and saving it locally and updating the file's metadata to reflect completed downloads.
- *terminate_connection:* Sends a CLOSE message to a peer or tracker and closes the associated socket.
- *request_file_from_tracker:* Interactively prompts the user to enter a filename and sends a request to the tracker. The tracker responds with a list of peers that have the file.

- *start_peer_operations:* Starts the peer's operations by connecting to the tracker and then begins listening for incoming connections from other peers.
- *refresh_available_files:* Requests the tracker for an updated list of files available for download from all peers.

4.2 Class Server

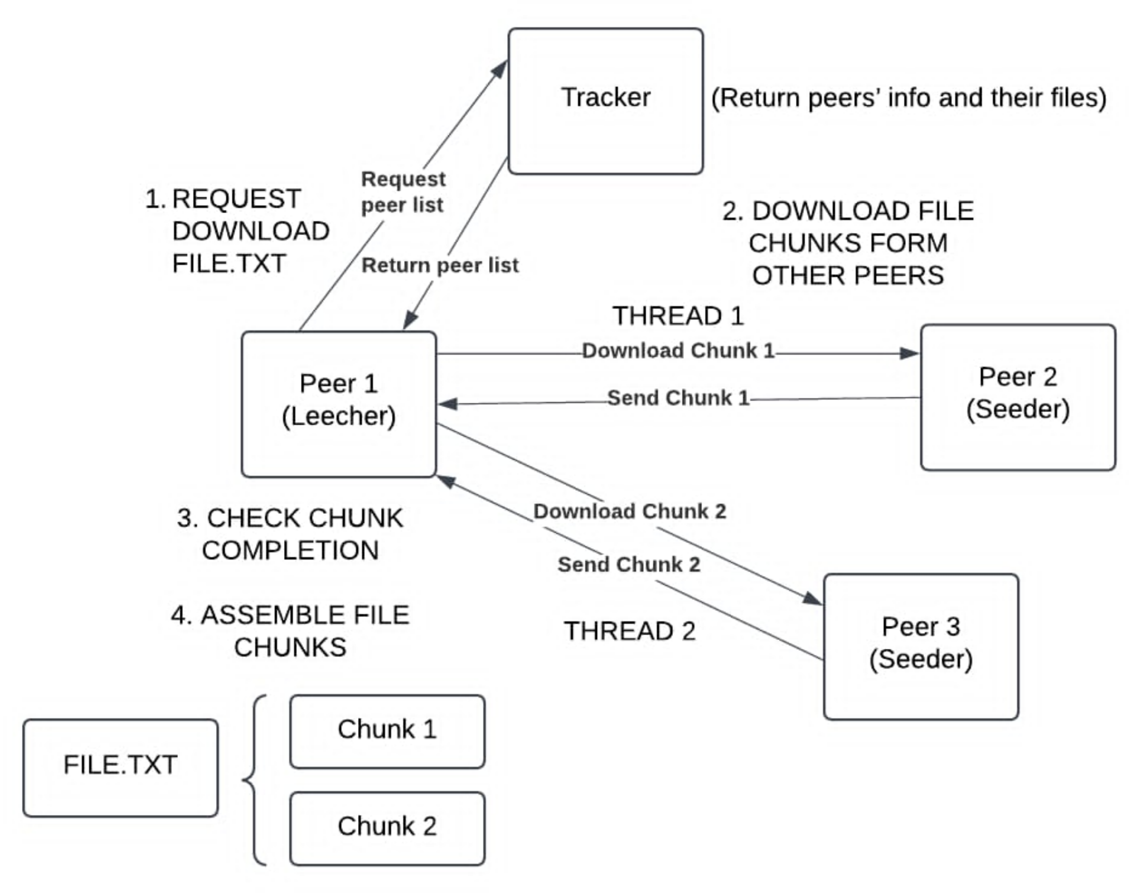
Manages communication with peers, tracks file availability across clients, and facilitates file-sharing requests by providing peer lists.

Below are the functions implemented for class Server:

- *__init__:* Initializes the tracker server with:
  - Its IP address and port.
  - A client_info dictionary to store client details and their files.
  - A files dictionary to map filenames to metadata.
  - A socket for listening to clients and a thread for handling client connections.
- *send_message_to_client:* Encodes and sends structured messages to clients based on the specified message type (HANDSHAKE, RESPONSE, etc.). Message include: Server ID (sid), File metadata and Peer lists if applicable.
- *decode_client_message:* Decodes incoming client messages into a structured dictionary. Handles various message types (e.g., HANDSHAKE, REQUEST) and extracts relevant information like file data and peer lists.
- *start_client_listener:* Continuously listens for incoming connections from peers using a timeout to prevent indefinite blocking. Creates a new thread for each client connection to handle requests concurrently.
- *update_peer_file_info:* Updates the tracker's records with the files and corresponding piece indexes available from a specific peer. Synchronizes access with a lock to ensure thread safety.

- *find_peers_with_file:* Searches for peers that have a specific file in their repository. Returns a list of peers along with their IP, port, and file piece indexes.
- *process_client_requests:* Processes requests from a specific peer, including:
  - **HANDSHAKE:** Adds the peer to client_info and updates the file list.
  - **REQUEST:** Finds peers that have the requested file and responds with peer details.
  - **CLOSE:** Removes the peer and its file information from the tracker.
  - **PEER_UPDATE_REQUEST:** Asks all peers for updates on file availability and responds with an updated file list.

5. Multi-Threaded Data Transfer (MDDT)



The diagram depicts the implementation of Multi-Threaded Data Transfer (MDDT) in the Simple Torrent-like Application (STA), demonstrating the interaction between the tracker and peers to download file chunks in parallel.

Step 1: Request File Information

- **Peer 1 (Leecher)** initiates a request to the **Tracker** to download FILE.TXT.
- The **Tracker** responds with a list of peers (Peer 2, Peer 3, etc.) that possess the required file chunks.

Step 2: Download File Chunks from Other Peers

- Upon receiving the peer list, **Peer 1** spawns multiple threads to download chunks in parallel:
    - **Thread 1** establishes a connection with **Peer 2 (Seeder)** to download **Chunk 1**.
    - **Thread 2** connects to **Peer 3 (Seeder)** to download **Chunk 2**.

Step 3: Check Chunk Completion

- Each thread independently manages its assigned chunk download and updates the shared file metadata upon completion.
- A synchronization mechanism (e.g., locks) ensures thread-safe updates, avoiding race conditions or redundant requests.

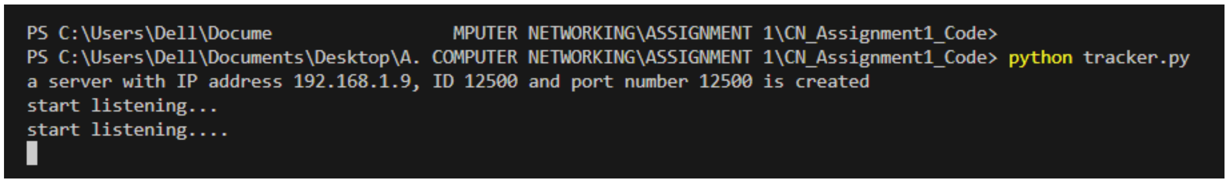
Step 4: Assemble File Chunks

- Once all chunks have been successfully downloaded, **Peer 1** combines **Chunk 1** and **Chunk 2** to reconstruct the complete file (FILE.TXT).

6. Application Manual

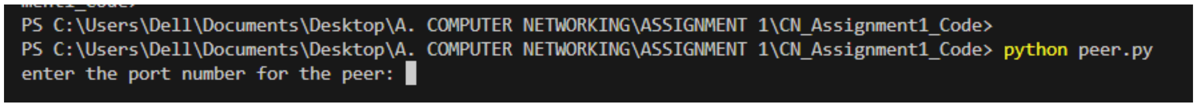
In this section, we provide a step-by-step manual to run the code.

Step 1: Run the tracker.py file



```
PS C:\Users\Dell\Docume                 MPUTER NETWORKING\ASSIGNMENT 1\CN_Assignment1_Code>
PS C:\Users\Dell\Documents\Desktop\A. COMPUTER NETWORKING\ASSIGNMENT 1\CN_Assignment1_Code> python tracker.py
a server with IP address 192.168.1.9, ID 12500 and port number 12500 is created
start listening...
start listening....
```

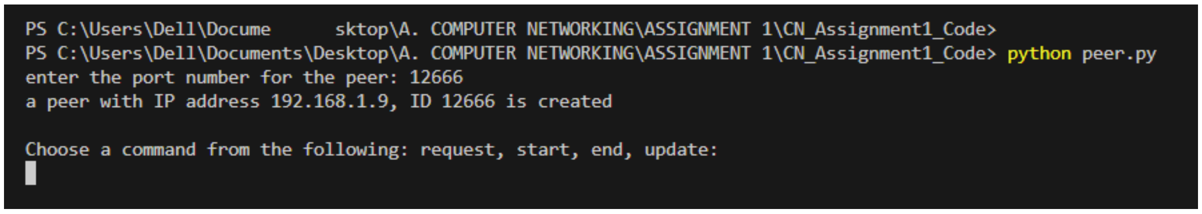
Step 2: Run the peer.py file in a different terminal. If many peers are involving, use each terminal for each peer.



```
PS C:\Users\Dell\Documents\Desktop\A. COMPUTER NETWORKING\ASSIGNMENT 1\CN_Assignment1_Code>
PS C:\Users\Dell\Documents\Desktop\A. COMPUTER NETWORKING\ASSIGNMENT 1\CN_Assignment1_Code> python peer.py
enter the port number for the peer:
```

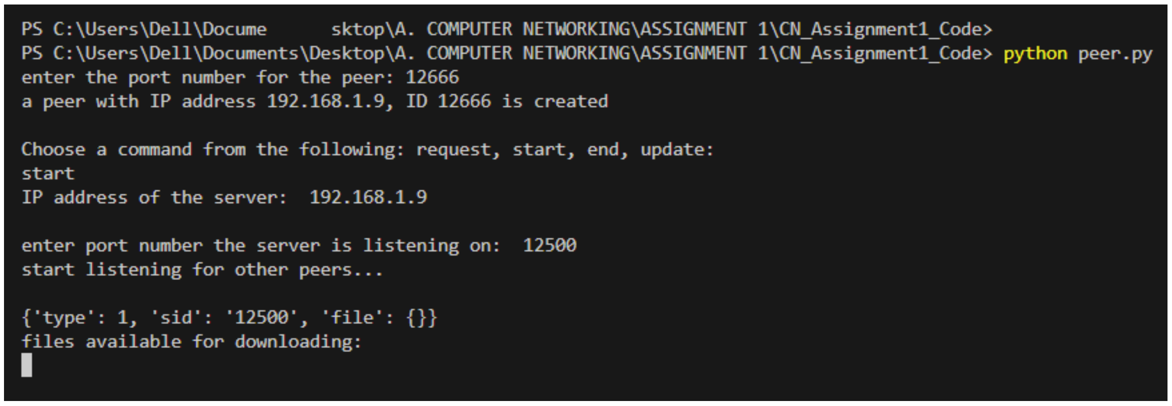
Figure 3. Run the tracker.py file

Step 3: For each peer, enter the start command to connect to the server and start listening to other peers.



```
PS C:\Users\Dell\Docume      sktop\A. COMPUTER NETWORKING\ASSIGNMENT 1\CN_Assignment1_Code>
PS C:\Users\Dell\Documents\Desktop\A. COMPUTER NETWORKING\ASSIGNMENT 1\CN_Assignment1_Code> python peer.py
enter the port number for the peer: 12666
a peer with IP address 192.168.1.9, ID 12666 is created

Choose a command from the following: request, start, end, update:
```

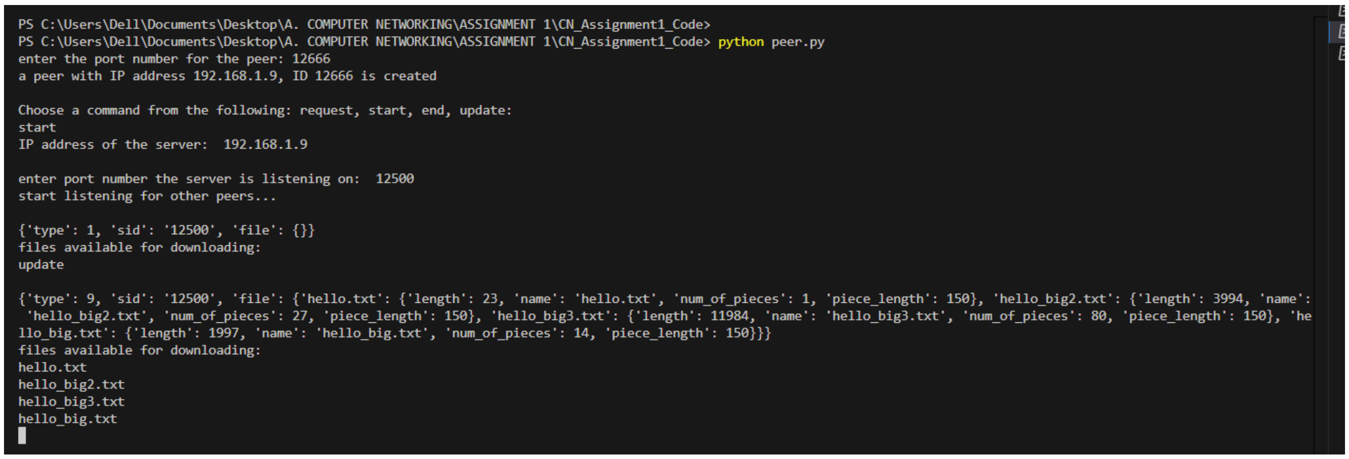
Figure 4. Run peer.py



```
PS C:\Users\Dell\Docume      sktop\A. COMPUTER NETWORKING\ASSIGNMENT 1\CN_Assignment1_Code>
PS C:\Users\Dell\Documents\Desktop\A. COMPUTER NETWORKING\ASSIGNMENT 1\CN_Assignment1_Code> python peer.py
enter the port number for the peer: 12666
a peer with IP address 192.168.1.9, ID 12666 is created

Choose a command from the following: request, start, end, update:
start
IP address of the server:  192.168.1.9

enter port number the server is listening on:  12500
start listening for other peers...

{'type': 1, 'sid': '12500', 'file': {}}
files available for downloading:
```

Figure 5. Enter "start" command

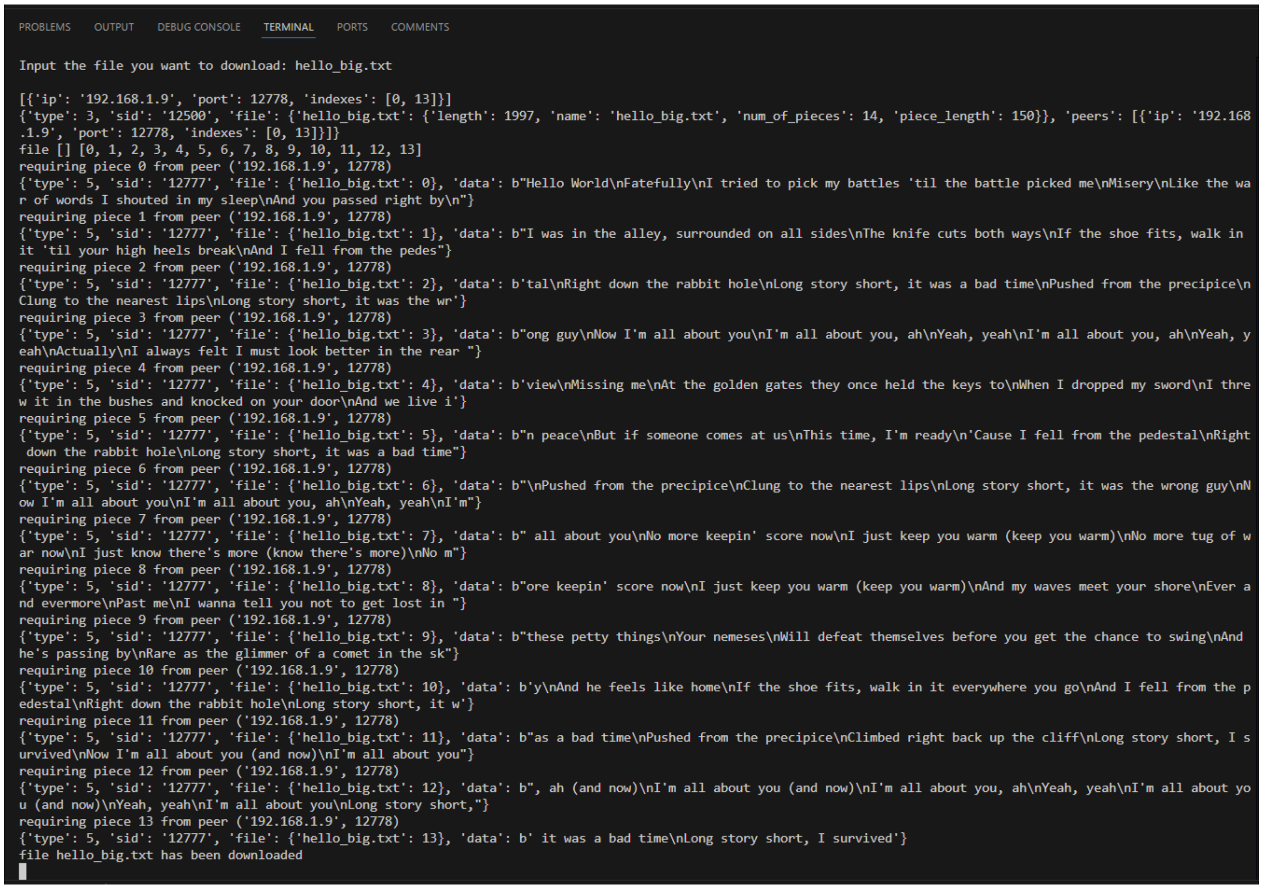
Step 4: (Important) Use the "update" command before "request" to get the most updated information.



```
PS C:\Users\Dell\Documents\Desktop\A. COMPUTER NETWORKING\ASSIGNMENT 1\CN_Assignment1_Code>
PS C:\Users\Dell\Documents\Desktop\A. COMPUTER NETWORKING\ASSIGNMENT 1\CN_Assignment1_Code> python peer.py
enter the port number for the peer: 12666
a peer with IP address 192.168.1.9, ID 12666 is created

Choose a command from the following: request, start, end, update:
start
IP address of the server:  192.168.1.9

enter port number the server is listening on:  12500
start listening for other peers...

{'type': 1, 'sid': '12500', 'file': {}}
files available for downloading:
update

{'type': 9, 'sid': '12500', 'file': {'hello.txt': {'length': 23, 'name': 'hello.txt', 'num_of_pieces': 1, 'piece_length': 150}, 'hello_big2.txt': {'length': 3994, 'name':
 'hello_big2.txt', 'num_of_pieces': 27, 'piece_length': 150}, 'hello_big3.txt': {'length': 11984, 'name': 'hello_big3.txt', 'num_of_pieces': 80, 'piece_length': 150}, 'he
llo_big.txt': {'length': 1997, 'name': 'hello_big.txt', 'num_of_pieces': 14, 'piece_length': 150}}}
files available for downloading:
hello.txt
hello_big2.txt
hello_big3.txt
hello_big.txt
```

In the figure above, we know that this peer does not have file hello_big.txt, so let's download the file.

Step 5: Download a file that is available for download. We use the "request" command, and then imput the file name we want to download, which is hello_big.txt.



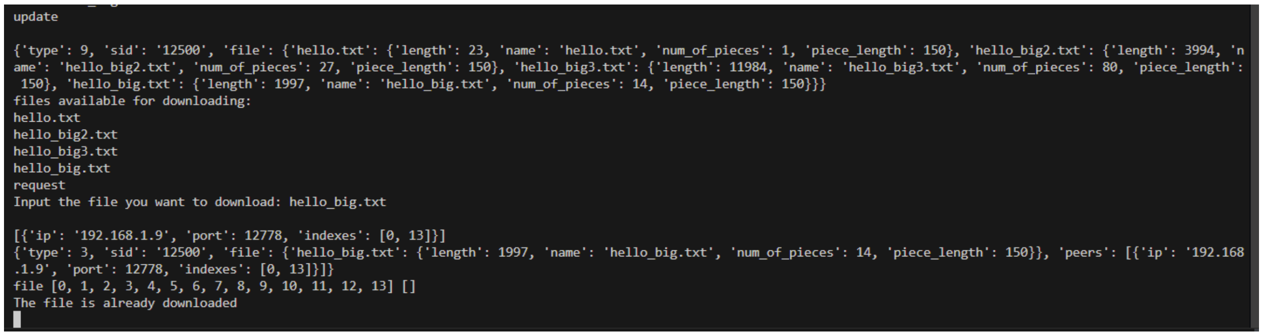
```
Input the file you want to download: hello_big.txt

[{'ip': '192.168.1.9', 'port': 12778, 'indexes': [0, 13]}]
{'type': 3, 'sid': '12500', 'file': {'hello_big.txt': {'length': 1997, 'name': 'hello_big.txt', 'num_of_pieces': 14, 'piece_length': 150}}, 'peers': [{'ip': '192.168
.1.9', 'port': 12778, 'indexes': [0, 13]}]}
file [] [0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13]
requiring piece 0 from peer ('192.168.1.9', 12778)
{'type': 5, 'sid': '12777', 'file': {'hello_big.txt': 0}, 'data': b"Hello World\nFatefully\nI tried to pick my battles 'til the battle picked me\nMisery\nLike the wa
r of words I shouted in my sleep\nAnd you passed right by\n"}
requiring piece 1 from peer ('192.168.1.9', 12778)
{'type': 5, 'sid': '12777', 'file': {'hello_big.txt': 1}, 'data': b"I was in the alley, surrounded on all sides\nThe knife cuts both ways\nIf the shoe fits, walk in
it 'til your high heels break\nAnd I fell from the pedes"}
requiring piece 2 from peer ('192.168.1.9', 12778)
{'type': 5, 'sid': '12777', 'file': {'hello_big.txt': 2}, 'data': b'tal\nRight down the rabbit hole\nLong story short, it was a bad time\nPushed from the precipice\n
Clung to the nearest lips\nLong story short, it was the wr'}
requiring piece 3 from peer ('192.168.1.9', 12778)
{'type': 5, 'sid': '12777', 'file': {'hello_big.txt': 3}, 'data': b"ong guy\nNow I'm all about you\nI'm all about you, ah\nYeah, yeah\nI'm all about you, ah\nYeah, y
eah\nActually\nI always felt I must look better in the rear "}
requiring piece 4 from peer ('192.168.1.9', 12778)
{'type': 5, 'sid': '12777', 'file': {'hello_big.txt': 4}, 'data': b'view\nMissing me\nAt the golden gates they once held the keys to\nWhen I dropped my sword\nI thre
w it in the bushes and knocked on your door\nAnd we live i'}
requiring piece 5 from peer ('192.168.1.9', 12778)
{'type': 5, 'sid': '12777', 'file': {'hello_big.txt': 5}, 'data': b"n peace\nBut if someone comes at us\nThis time, I'm ready\n'Cause I fell from the pedestal\nRight
 down the rabbit hole\nLong story short, it was a bad time"}
requiring piece 6 from peer ('192.168.1.9', 12778)
{'type': 5, 'sid': '12777', 'file': {'hello_big.txt': 6}, 'data': b"\nPushed from the precipice\nClung to the nearest lips\nLong story short, it was the wrong guy\nN
ow I'm all about you\nI'm all about you, ah\nYeah, yeah\nI'm"}
requiring piece 7 from peer ('192.168.1.9', 12778)
{'type': 5, 'sid': '12777', 'file': {'hello_big.txt': 7}, 'data': b" all about you\nNo more keepin' score now\nI just keep you warm (keep you warm)\nNo more tug of w
ar now\nI just know there's more (know there's more)\nNo m"}
requiring piece 8 from peer ('192.168.1.9', 12778)
{'type': 5, 'sid': '12777', 'file': {'hello_big.txt': 8}, 'data': b"ore keepin' score now\nI just keep you warm (keep you warm)\nAnd my waves meet your shore\nEver a
nd evermore\nPast me\nI wanna tell you not to get lost in "}
requiring piece 9 from peer ('192.168.1.9', 12778)
{'type': 5, 'sid': '12777', 'file': {'hello_big.txt': 9}, 'data': b"these petty things\nYour nemeses\nWill defeat themselves before you get the chance to swing\nAnd
he's passing by\nRare as the glimmer of a comet in the sk"}
requiring piece 10 from peer ('192.168.1.9', 12778)
{'type': 5, 'sid': '12777', 'file': {'hello_big.txt': 10}, 'data': b'y\nAnd he feels like home\nIf the shoe fits, walk in it everywhere you go\nAnd I fell from the p
edestal\nRight down the rabbit hole\nLong story short, it w'}
requiring piece 11 from peer ('192.168.1.9', 12778)
{'type': 5, 'sid': '12777', 'file': {'hello_big.txt': 11}, 'data': b"as a bad time\nPushed from the precipice\nClimbed right back up the cliff\nLong story short, I s
urvived\nNow I'm all about you (and now)\nI'm all about you"}
requiring piece 12 from peer ('192.168.1.9', 12778)
{'type': 5, 'sid': '12777', 'file': {'hello_big.txt': 12}, 'data': b", ah (and now)\nI'm all about you (and now)\nI'm all about you, ah\nYeah, yeah\nI'm all about yo
u (and now)\nYeah, yeah\nI'm all about you\nLong story short,"}
requiring piece 13 from peer ('192.168.1.9', 12778)
{'type': 5, 'sid': '12777', 'file': {'hello_big.txt': 13}, 'data': b' it was a bad time\nLong story short, I survived'}
file hello_big.txt has been downloaded
```

This is the final result when the download is finished.

***What if a peer download the file that already has?**

Let's download the file hello_big.txt again and see what will happen.

We use the "update" command to get the most updated information, and then request to download the file hello_big.txt again.



```
update

{'type': 9, 'sid': '12500', 'file': {'hello.txt': {'length': 23, 'name': 'hello.txt', 'num_of_pieces': 1, 'piece_length': 150}, 'hello_big2.txt': {'length': 3994, 'n
ame': 'hello_big2.txt', 'num_of_pieces': 27, 'piece_length': 150}, 'hello_big3.txt': {'length': 11984, 'name': 'hello_big3.txt', 'num_of_pieces': 80, 'piece_length':
 150}, 'hello_big.txt': {'length': 1997, 'name': 'hello_big.txt', 'num_of_pieces': 14, 'piece_length': 150}}}
files available for downloading:
hello.txt
hello_big2.txt
hello_big3.txt
hello_big.txt
request
Input the file you want to download: hello_big.txt

[{'ip': '192.168.1.9', 'port': 12778, 'indexes': [0, 13]}]
{'type': 3, 'sid': '12500', 'file': {'hello_big.txt': {'length': 1997, 'name': 'hello_big.txt', 'num_of_pieces': 14, 'piece_length': 150}}, 'peers': [{'ip': '192.168
.1.9', 'port': 12778, 'indexes': [0, 13]}]}
file [0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13] []
The file is already downloaded
```

As we can see from the figure above, a peer will be informed if he/she downloads a file that already exists.

7. Conclusion

The Simple Torrent-like Application (STA) developed as part of this project successfully demonstrates a peer-to-peer file-sharing system, leveraging the TCP/IP protocol stack. Through the collaboration of a centralized tracker and multiple peers, the application achieves efficient and reliable file distribution across the network. The implementation showcases the following key features:

1. **Multi-Directional Data Transfer (MDDT):** Peers can download file pieces simultaneously from multiple sources, optimizing network bandwidth and reducing download times.
2. **Scalability:** The tracker and peer architecture supports dynamic addition or removal of peers, making the system robust and adaptable to real-world network conditions.
3. **Interactive Functionality:** Commands for file requests, updates, and termination provide users with control over the application, enhancing usability.

The project serves as a practical application of networking concepts, including socket programming, multithreading, and distributed file management. By implementing both tracker and peer functionalities, the STA highlights the potential of decentralized systems for efficient data sharing.