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

Computer Network

Assignment 1

DEVELOP A NETWORK APPLICATION

Advisor(s):	Diep Thanh Dang	
Student(s):	Luu Chi Cuong	2252097
	Nguyen Huu Cuong	2252098
	Nguyen Trinh Chau	2252091

HO CHI MINH CITY, DECEMBER 2024

Contribution Table		
ID	**Name**	**Contribution**
2252097	Luu Chi Cuong	33%
2252098	Nguyen Huu Cuong	34%
2252091	Nguyen Trinh Chau	33%

Contents

1 Introduction

1.1 Peer to peer network

In modern networking, peer-to-peer (P2P) file-sharing systems have become an essential application model, enabling efficient and decentralized data distribution. In a P2P network, computers directly connect and share resources without a central server.

Within a peer-to-peer (P2P) network, every node or connection—such as a router, printer, switch, or computer—connects directly with one another to share resources, such as files, data, or processing power. It has a decentralized network architecture, meaning the nodes connect across the network without a centralized server.

P2P networks are useful for applications that require decentralized collaboration, resource sharing, or secure and transparent transactions. However, P2P networks may not be suitable for applications that require centralized control, high levels of security, or ones in which legal issues are a concern.

This report presents the development of a Simple Torrent-like Application (STA) designed to emulate the functionality of popular file-sharing protocols such as BitTorrent. The application leverages the TCP/IP protocol stack and supports multi-directional data transferring (MDDT), allowing for simultaneous downloads from multiple sources.

1.2 Difference between Peer-to-Peer and Client-Server architecture

The Client-Server model are broadly used in network model. In this architecture, Clients and servers are differentiated, and Specific servers and clients are present. In Client-Server Network, a Centralized server is used to store the data because its management is centralized. In Client-Server Network, the Server responds to the services which is requested by the Client.

Main differences

Client-Server Model

- In Client-Server Network, Clients and server are differentiated, Specific server and clients are present.
- In Client-Server Network, Centralized server is used to store the data.

- In Client-Server Network, Server respond the services which is request by Client.
- Client-Server Network are more stable than Peer-to-Peer Network.
- Client-Server Network is used for both small and large networks.

Peer-to-Peer Model

- In Peer-to-Peer Network, Clients and server are not differentiated.
- While in Peer-to-Peer Network, Each peer has its own data.
- While in Peer-to-Peer Network, Each and every node can do both request and respond for the services.
- While Peer-to-Peer Network are less stable if number of peer is increase.
- While Peer-to-Peer Network is generally suited for small networks with fewer than 10 computers.

1.3 Bit Torrent Introduction

In this Assignment, we will implement a Simple Torrent-like Application (STA). BitTorrent, also referred to simply as torrent, is a communication protocol for peer-to-peer file sharing (P2P), which enables users to distribute data and electronic files over the Internet in a decentralized manner. The protocol is developed and maintained by Rainberry, Inc., and was first released in 2001.

The BitTorrent protocol can be used to reduce the server and network impact of distributing large files. Rather than downloading a file from a single source server, the BitTorrent protocol allows users to join a "swarm" of hosts to upload or download from each other simultaneously.

Several basic computers can replace large servers while efficiently distributing files to many recipients using the BitTorrent protocol. In addition, this lower bandwidth helps prevent large spikes in internet traffic in a given area, keeping internet speeds higher for all users, regardless of whether or not they use the BitTorrent protocol.

However, Bit Torrent is not a completely P2P network. It leverages the advantage of a client-server network by deploying a tracker. It is server software that coordinates the transfer of files among users. The tracker does not contain a copy of the file and helps peers discover each other. The tracker and the client exchange information using a simple protocol TCP/IP. The clients inform the tracker regarding the file they want to download, their IP, and port, and the tracker responds with a list of peers downloading the same file and their contact information. This list of peers that all share the same Torrent represents a swarm. The tracker is necessary for peers to find each other and transfer data. Because of the presence of this central entity, BitTorrent protocol is considered a Hybrid P2P implementation.

1.4 Multi-Directional Data Transferring (MDDT)

MDDT enables simultaneous downloading of pieces from multiple nodes. This feature maximizes network bandwidth utilization and minimizes download time. We implement MDDT using multi-threading. Also, a peer can also both download and upload at the same time.

To achieve MDDT, we employ multi-threading as the core implementation strategy. Each download or upload request from a peer is assigned to a dedicated thread, enabling concurrent operations without blocking other network activities. This threading model ensures that each connection operates independently, minimizing latency and optimizing resource usage.

The multi-threading approach is not limited to peer connections alone; it also extends to the tracker, a central component that manages connections and coordination between peers. The tracker employs a pool of threads to handle multiple client requests concurrently. This enables the tracker to efficiently process operations such as peer registration, file piece availability updates, and the provision of peer lists to requesting clients. By allowing the tracker to handle multiple client connections simultaneously, the system ensures that bottlenecks at the central coordination point are minimized.

One of the key advantages of MDDT is its ability to support parallel file downloads and uploads. A client can initiate multiple threads to download pieces of a single file from different source nodes. Similarly, it can simultaneously request pieces of other files, enabling multi-file downloads. This parallelism not only accelerates the download process but also ensures better fault tolerance, as the system can dynamically redistribute tasks if a particular node becomes unavailable.

1.5 Key Components:

- **Tracker**: A centralized server that maintains metadata, such as the availability of files and pieces and the list of peers that hold those pieces. It coordinates between peers and aids in locating resources.
- **Peer**: Clients that share or download file segments from other nodes in the network. Peer maintain repositories of available pieces and communicate with the tracker to download or upload data.
- **Pieces**: Files are divided into smaller segments of fixed size, in this assignment we choose the size of 512Kb. This segmentation enables efficient parallel downloading and error handling.
- **Metainfo File**: Known as a .torrent file, holds all the details about the torrent, including where the tracker address (IP address) is, what is the piece length, and piece-count. It can be considered as the info where magnet text points to and is combined with the current list of trackers and nodes.

2 Communication Protocols

2.1 Tracker HTTP Protocol

First, a peer connect to the tracker on a permanent port. Then, it send the request, specify what is the service the peer need from the tracker.

Port specification

Before a peer can use a service from the tracker, it must specify its port number so that the tracker knows where to send back the response.

The process can be done by sending a message to the tracker to tell which port client is using. If no error occurs, the tracker acknowledge the message and they start to communicate.

Downloading

If a peer need to download a file, it must send the to the tracker, specify the file they want to download. If the tracker has a torrent file of the requested file, it will send back the torrent file.

The tracker then search in its peer dictionary to check if any peer has that file. If some peers do, the tracker response with a list of peer (with ip address and port number) and the pieces that peer holds. Otherwise, the tracker return the message "No peer has the file".

Uploading

If a peer has a file and they want to share it with other peers. They need to send the Torrent File that contain the information of that file to the tracker.

The tracker check its own directory and if the file in the torrent is not yet saved by the tracker, it will save it and broadcast to all peer the new file that are ready to download.

Updating

After a peer had downloaded some pieces, it can update to the tracker the new pieces it held at that moment. And the tracker broadcast it to every peer.

2.2 Peer-to-Peer Protocol:

Peers connect to each other based on information from the tracker. Peers exchange information about the pieces they own and request missing pieces from others.

Peer Downloading

For a peer to request a piece from another peer, it first need to get the IP address and port number of the peers that hold that piece from the tracker. After that the requesting peer will connect to the requested peer, then they establish a connection and start exchanging pieces.

The requested peer will send all pieces it has for that file to the requesting peer. The requesting peer calculates its missing pieces.

For each missing piece of the requesting peer that are available in the requested piece. It starts to download that piece. After downloading that piece, it will update the piece it has now and also update it on the tracker.

Peer Uploading

To upload a file or some pieces of the file, the protocol is pretty much a reverse version of Downloading.

First the peer that want to upload, in this part we call it upload piece, send a message to the download peer. After receive an acknowledgement from the download peer, the upload peer send the file name and piece, as well as the piece length. You will see it's pretty much the same with Downloading. Then the upload peer will receive a message "READY" indicating that the downloading peer now ready to download. Then it start to send the piece to the download peer. When the download peer successfully download the piece, it return the message "Finish" and the process ends.

2.3 Detailed application design

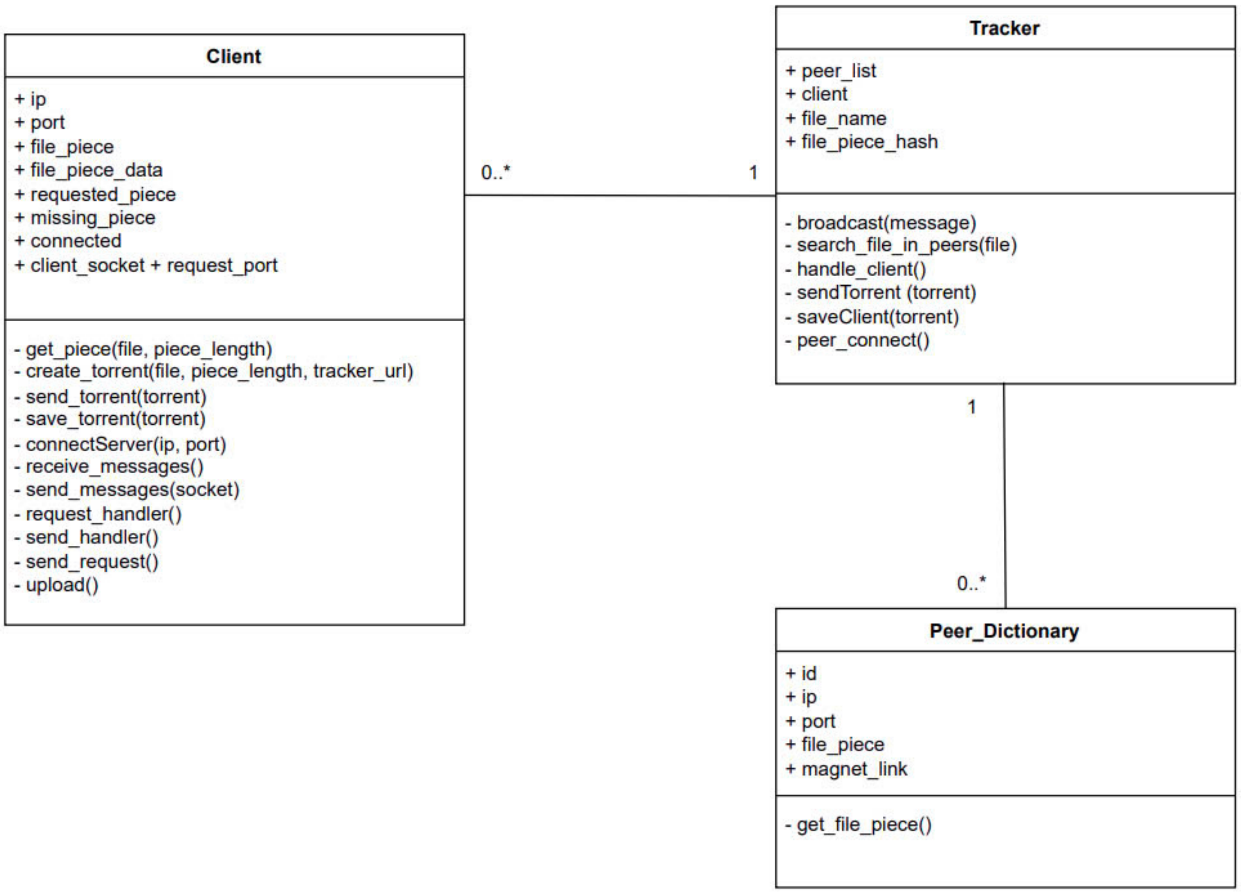


Figure 1: Use case diagram

This class diagram represents the architecture of a simplified file-sharing application with the main components and their relationships.

2.3.1 Client

Attribute:

- ip, port: Identifies the client on the network.
- file_piece: Stores information about file pieces.
- requested_piece, missing_piece: Tracks requested and missing file pieces.
- connected: List of connected peers.
- client_socket, request_port: Handles networking for communication with peers.

Methods:

- get_piece(file, piece_length): Splits a file into pieces.
- create_torrent(file, piece_length, tracker_url): Creates a torrent file for sharing.
- send_torrent(torrent): Sends the torrent to other peers or the tracker.
- save_torrent(torrent): Saves the torrent file locally.
- connectServer(ip, port): Establishes a connection with a peer.
- request_handler(), send_request(): Handles requests for file pieces.
- upload(): Uploads file pieces to other peers.

2.3.2 Tracker

Attribute

- peer_list: Maintains a list of all registered peers.
- client, file_name: Tracks client information and associated files.
- file_piece_hash: Stores hash values of file pieces for integrity checks.

Methods:

- broadcast(message): Sends messages to all connected peers.
- handle_client(client): Manages incoming connections from clients.

- sendTorrent(torrent): Shares the torrent file with peers.
- saveClient(torrent): Registers a client with the tracker.
- peer_connect(): Establishes connections between peers.

2.3.3 Peer_Dictionary

Attribute

- id: Unique identifier for the peer.
- ip, port: Network details of the peer.
- file_piece: Information about the file pieces the peer holds.

2.3.4 Relationships

- **Client and Tracker**: A many-to-one relationship. Multiple clients connect to the tracker for coordination and to retrieve information about available peers and files.
- **Tracker to Peer_Dictionary**: A one-to-many relationship. The tracker maintains metadata about multiple peers using the Peer_Dictionary. Each peer information is saved in a Peer_Dictionary.

3 Implementation

3.1 Peer Function Specification

3.1.1 Function: get_pieces(file, length)

Purpose: This function breaks down a file into multiple pieces with a fix length
Inputs:

- file: A file that you want to get pieces
- length: The piece length you want to get for each piece.

Output: A list of hashed pieces with fixed length except the last piece.
Process:

- Open a folder to save the pieces

- Read each chunk of length and save it in the folder
- Update the list of piece
- Hash the piece and append it to the return list.

3.1.2 Function: create_torrent(file, length, url)

Purpose: This function read a file and create a torrent file.
Inputs:

- file: A file that you want to make the torrent file
- length: The piece length specify of each piece.
- tracker_url: The link to tracker embedded in the torrent file.

Output: The torrent file is saved in the client folder.
Process:

- Call the get_piece() function to have the pieces from the file
- Append tracker url, file_name, piece, piece_length to a torrent file
- Call the save_torrent() function to save the torrent file

3.1.3 Function: send_torrent(torrent_path, socket)

Purpose: This function is use to send the torrent file to the tracker
Inputs:

- torrent_path: the path to the torrent file
- socket: The destination to send to

Output: The torrent is sent to tracker.
Process:

- Open the torrent file using the file path
- Read torrent file and send it via the socket to the destination socket.

3.1.4 Function: connectServer()

Purpose: This function is use to connect to the tracker and initialize the communication.

Inputs: null

Output: Connection is establish

Process:

- Start a new thread to receive the request from other peer.
- Create a socket and connect to Tracker
- Send the port number, the file peer holds and receive a peer list from tracker.
- Start 2 new threads for: receive message and send message.

3.1.5 Function: send_message(socket)

Purpose: Handles communication from the client side in a peer-to-peer system. The function listens for user inputs, processes commands and request to the tracker.

Inputs:

- client_socket: A socket to connect to the server

Output: A torrent file is received and missing pieces are requested

Process:

- Wait for user to enter a message
- If the message is exit, close the connection
- If the message is Download, start the download process. First, Peer requests for torrent file if that torrent is not found. Then it calculates the missing piece and start a thread to request those pieces.
- If the message is Upload, it first connects to the tracker to get the active peer list. Then it start to upload to those peer.

3.1.6 Function: receive_message(socket)

Purpose: Handles incoming messages from the server.

Inputs:

- client_socket: A socket to receive messages from the server

Output: None

Process:

- Create a loop to receive from the Tracker
- Ìf the message start with File, it is the broadcast message from tracker in case there is a change in the available file list, such as there are some new files or there are some files that are removed.
- If the message is RESPOND TORRENT, decode the torrent file, calculate missing pieces and start a thread to request for those pieces.
- If the message is RESPOND DOWNLOAD, receive the list of peer hold that file.
- If the message is RESPOND PIECE, receive the list of peer (IP address and port) hold that piece.
- If the message is RESPOND PEER, receive the list of peer in the system except themselves which are used to upload piece to.

3.1.7 Function: request_handler(socket)

Purpose: Handles incoming connection requests from peers in a peer-to-peer system. The function listens for connections and forwards handling of these requests to a separate handler function.

Inputs: None

Output: Peer request is handled

Process:

- Create socket to receive the request from other peers
- Accept the connection
- Call send_handler to send pieces for the requesting peer.

3.1.8 Function: send_handler(client_socket)

Purpose: Handles file transfer requests (upload and download) between peers in a peer-to-peer system. This function processes initial client requests, manages file piece transfers, and ensures the integrity of transmitted data.

Inputs: client_socket: A socket to send and receive message.

Output: Peer successfully upload or download the file

Process:

- If the received message is "down", peer search for that file, if exist, it send a message "start" and if the requesting peer are ready, it first send the piece size and after receiving an acknowledgement, it start to send the piece.
- If the message is "up", it first make a directory to save that piece. Then it start to download the piece until the size reach the piece length. Then it update to the tracker.

3.1.9 Function: send_request()

Purpose: Responsible the downloading of missing file pieces in a peer-to-peer network by identifying missing pieces, connecting to peers, and requesting the data for those pieces.

Inputs: None

Output: Missing pieces are request to other peers and those pieces are successfully downloaded.

Process:

- First it connects to the tracker to get the peer list holding that piece.
- If there is any peer that has the piece but has not yet been connect, connect to that peer.
- It start to initialize the communicaiton by sending "down" and then "file_name" messages,
- The peer receive the list of pieces that is available by the requested peer. And start to request the piece it needs.
- After ack "READY", it start to download that piece. Updating the list of missing piece and request piece. It also update the piece list it holds to the tracker.

3.1.10 Function: upload(list_peer)

Purpose: Responsible the uploading random pieces to the peers

Inputs: list_peer: the peer list that we want to upload to.

Output: Successfully upload to other peers

Process:

- The process is quite similar to downloading but in a reverse manner.
- First choose a random piece but not exceed the available pieces.
- After initializing the connection and receiving the acknowledgment "READY", it start to upload the piece.

3.2 Tracker Function Specification

3.2.1 Function: broadcast(message)

Purpose: Responsible for broadcasting the message to all peers.
Inputs: message: the message that tracker want to broadcast
Output: Message is broadcasted to all peers.
Process:

- For all clients in the client list, the tracker send the message to that peer.

3.2.2 Function: removeInfo(peer)

Purpose: When a peer disconnect to the system. The tracker need to update the list of available pieces
Inputs: peer: the peer that is to remove
Output:Peer is remove and the available list is updated.
Process:

- First the tracker remove the target peer and update the new piece list.
- Then it check all the torrent file and calculate the missing piece.
- If any file is not complete because of the absence of that peer, remove that file.

3.2.3 Function: request_handler(address)

Purpose: Handles incoming client requests, updates peer information, and facilitates file-sharing operations.
Inputs: address: IP address and port number of the client
Output: Client is handled
Process:

- If receive the message TORRENT: Sends a torrent file for a requested filename.

- If receive the message FILE: Updates file_piece_hash with the pieces of a file available on the peer.
- If receive the message PORT: Updates the peer's port information.
- If receive the message UPDATE FILE: Updates file piece availability in file_piece_hash.
- If receive the message SEND: Receives and saves torrent metadata, then broadcasts the updated file list to all peers.
- If receive the message PIECE: Retrieves and sends details of peers holding a specific file piece.
- If receive the message PEER: Sends a list of all connected peers, excluding the requesting peer.

3.2.4 Function: peer_connect()

Purpose: To set up a server that listens for incoming peer connections and spawns a dedicated thread for each connection.

Inputs: None

Output: Successfully establish a connection and create a new thread to handle client.

Process:

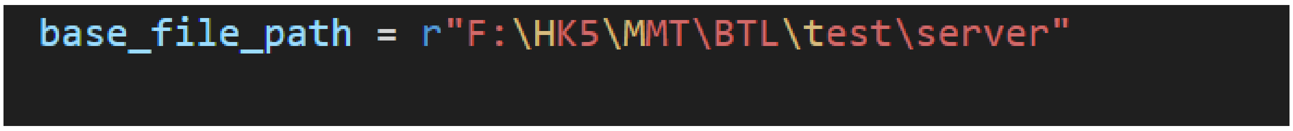
- Creates a TCP socket and Binds the socket to IP '0.0.0.0' (all available interfaces) and port 5000.
- Puts the socket in listening mode
- If thers is a connection request ,accepts incoming client connections to the self.clients list to track connected peers.
- Creates a new thread to handle each connected client. Keeps the server running continuously to accept multiple client connections.

4 Result

4.1 Sanity test

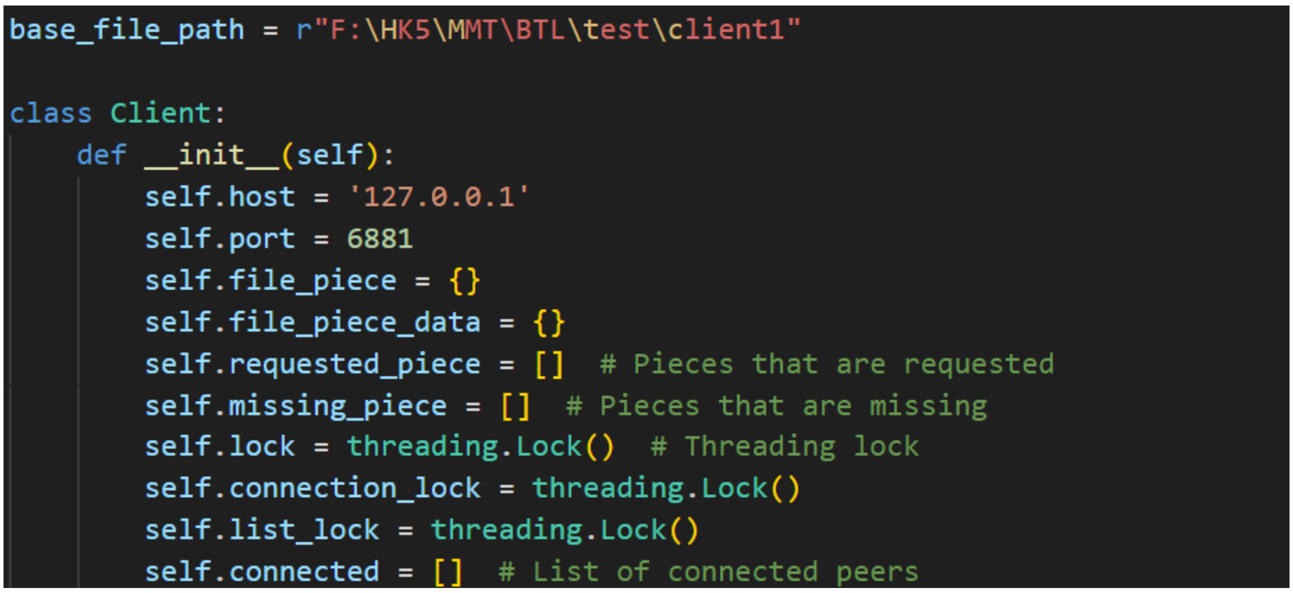
4.1.1 Environment and configuration set up

- For server: Choose a folder to store the received torrent file into and save the address in Code.



```
base_file_path = r"F:\HK5\MMT\BTL\test\server"
```

- For client: Choose your folder and put the file you want to share into it. Save folder address you use to share in Code and set up the listening port of that client.



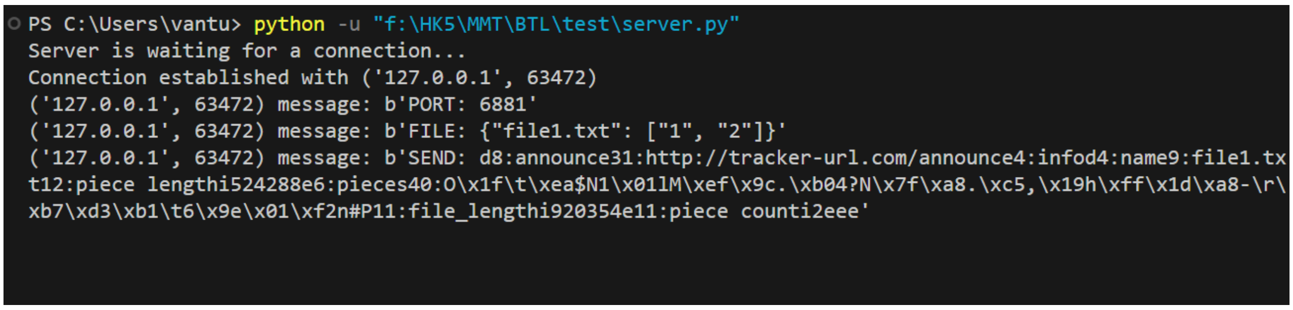
```
base_file_path = r"F:\HK5\MMT\BTL\test\client1"

class Client:
    def __init__(self):
        self.host = '127.0.0.1'
        self.port = 6881
        self.file_piece = {}
        self.file_piece_data = {}
        self.requested_piece = []  # Pieces that are requested
        self.missing_piece = []  # Pieces that are missing
        self.lock = threading.Lock()  # Threading lock
        self.connection_lock = threading.Lock()
        self.list_lock = threading.Lock()
        self.connected = []  # List of connected peers
```

The syntax for downloading and uploading:

- DOWNLOAD: Syntax: DOWNLOAD: <file_name>
- UPLOAD: Syntax: UPLOAD: <file_name>

4.1.2 Tracker connecting

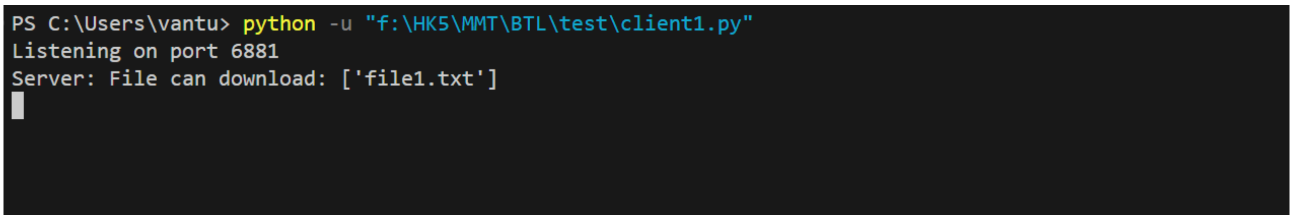
First, to launch the application, we first need to start the tracker.



```
PS C:\Users\vantu> python -u "f:\HK5\MMT\BTL\test\server.py"
Server is waiting for a connection...
Connection established with ('127.0.0.1', 63472)
('127.0.0.1', 63472) message: b'PORT: 6881'
('127.0.0.1', 63472) message: b'FILE: {"file1.txt": ["1", "2"]}'
('127.0.0.1', 63472) message: b'SEND: d8:announce31:http://tracker-url.com/announce4:infod4:name9:file1.tx
t12:piece lengthi524288e6:pieces40:O\x1f\t\xea$N1\x01lM\xef\x9c.\xb04?N\x7f\xa8.\xc5,\x19h\xff\x1d\xa8-\r\
xb7\xd3\xb1\t6\x9e\x01\xf2n#P11:file_lengthi920354e11:piece counti2eee'
```

Figure 4.1: Tracker launched

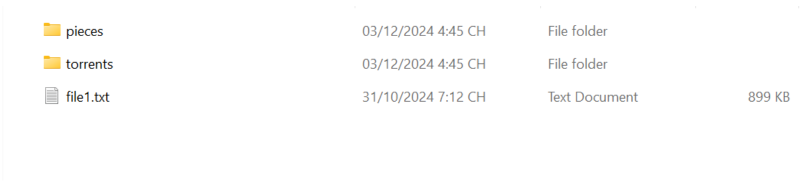
After being started and listening on port 5000. The tracker has an connection request from the first client and establish a connection.



```
PS C:\Users\vantu> python -u "f:\HK5\MMT\BTL\test\client1.py"
Listening on port 6881
Server: File can download: ['file1.txt']
```

Figure 4.2: Client launch and connect to tracker

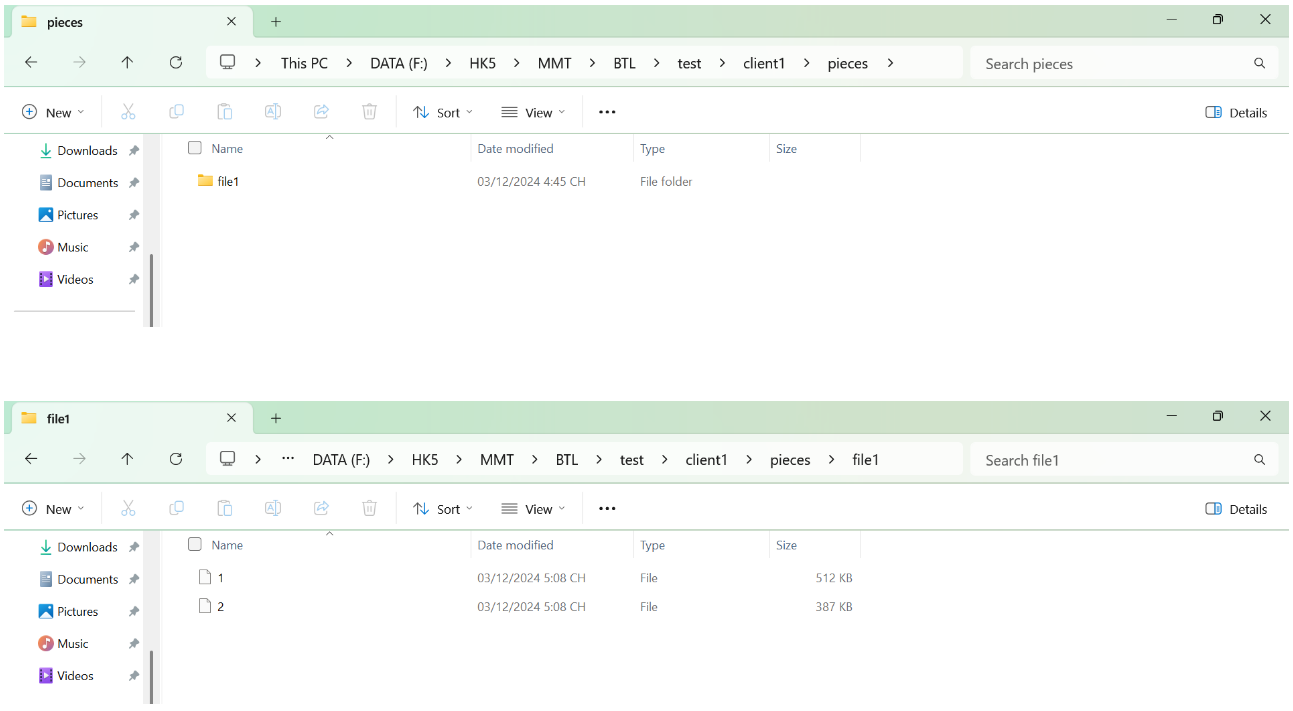
The client sends its port number, the files it holds and the torrent for that file to server which result in the tracker terminal in the picture 4.1. The tracker then broadcasts to all users the new available set of files.



pieces	03/12/2024 4:45 CH	File folder	
torrents	03/12/2024 4:45 CH	File folder	
file1.txt	31/10/2024 7:12 CH	Text Document	899 KB

Figure 4.3: Pieces and Torrents are made and save in 2 folder.

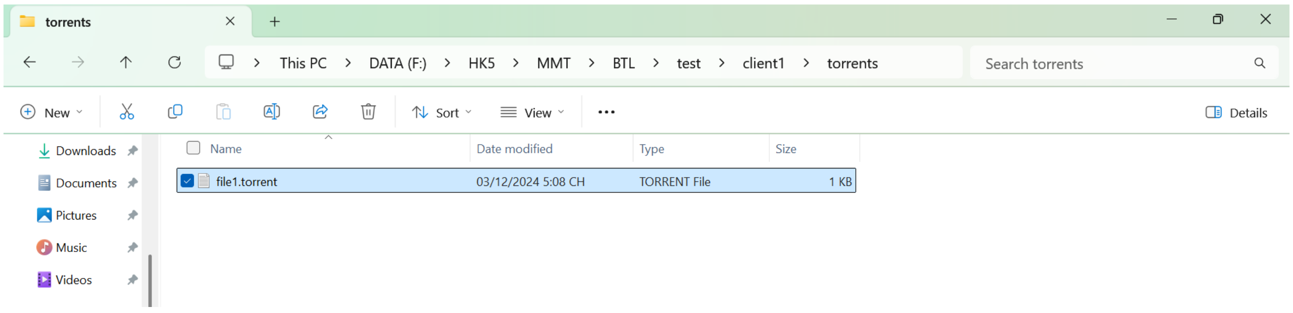
In order to send torrent to the tracker and start to share files to other peer. The client application first need to divide the file into pieces. Then the piece is hashed and saved into the torrent file. After this process, the pieces and torrent are save into the folder in the host device.


pieces
This PC > DATA (F:) > HK5 > MMT > BTL > test > client1 > pieces >
Search pieces
New
Sort
View
Details
Downloads
Documents
Pictures
Music
Videos
file1
… > DATA (F:) > HK5 > MMT > BTL > test > client1 > pieces > file1
Search file1

Name	Date modified	Type	Size
file1	03/12/2024 4:45 CH	File folder	

Name	Date modified	Type	Size
1	03/12/2024 5:08 CH	File	512 KB
2	03/12/2024 5:08 CH	File	387 KB

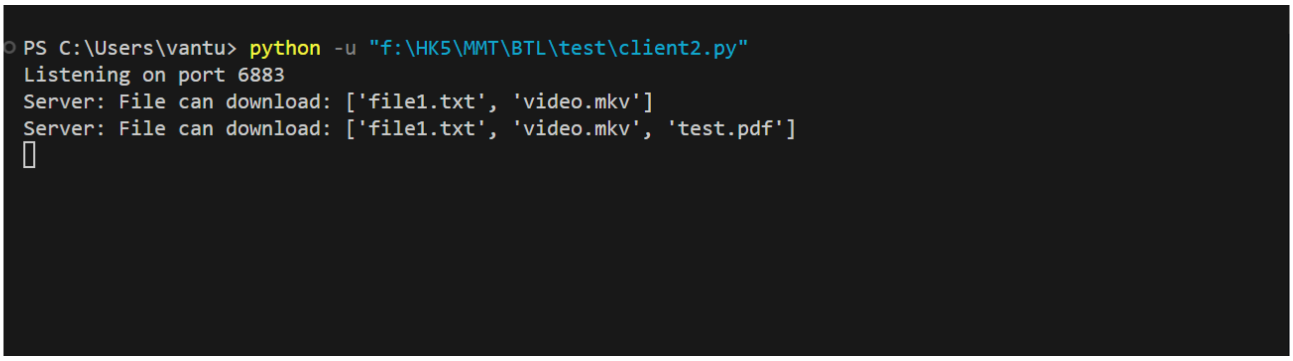
Figure 4.4: Pieces folder.



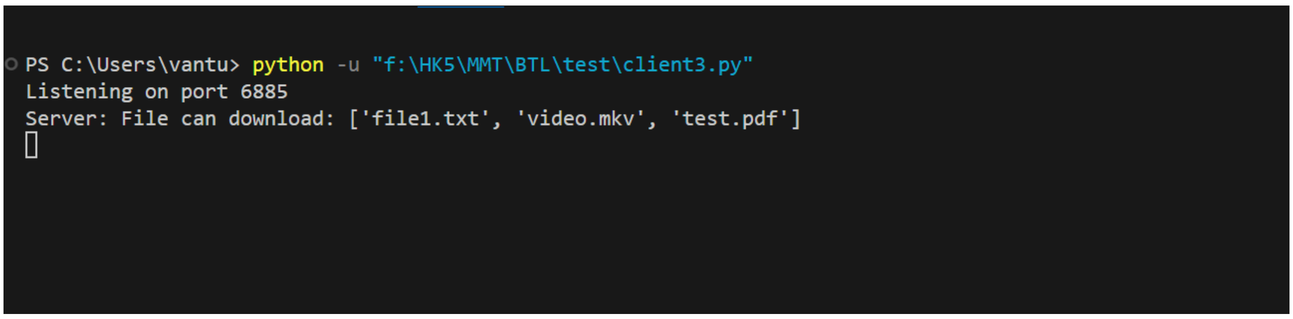
Name	Date modified	Type	Size
file1.torrent	03/12/2024 5:08 CH	TORRENT File	1 KB

Figure 4.5: Torrent folder

Then if a new peer comes, it will provide more available files to share. The tracker will broadcast to all peers if there is a change in available file list.



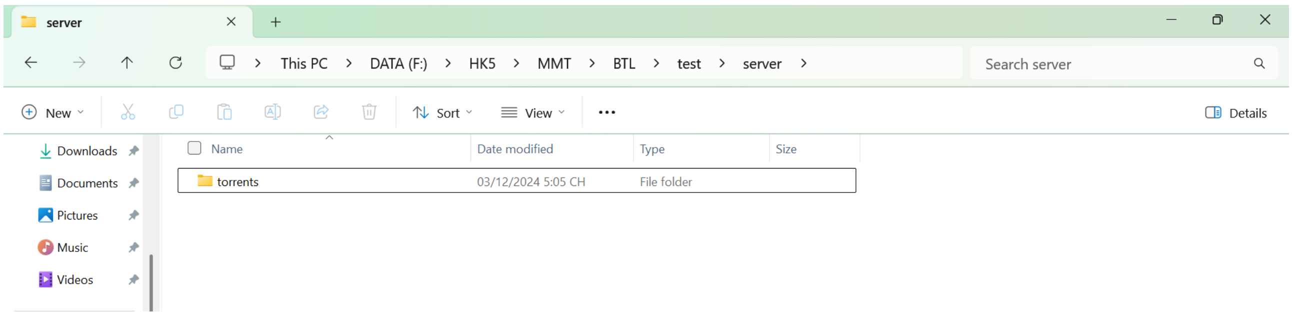
```
PS C:\Users\vantu> python -u "f:\HK5\MMT\BTL\test\client2.py"
Listening on port 6883
Server: File can download: ['file1.txt', 'video.mkv']
Server: File can download: ['file1.txt', 'video.mkv', 'test.pdf']
```



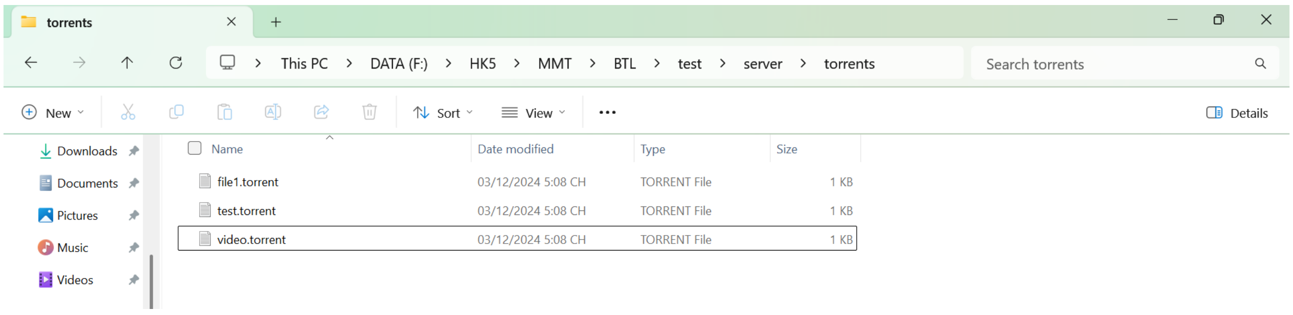
```
PS C:\Users\vantu> python -u "f:\HK5\MMT\BTL\test\client3.py"
Listening on port 6885
Server: File can download: ['file1.txt', 'video.mkv', 'test.pdf']
```

Figure 4.6: New peer provide new file.

When new a new peer joins, they send the torrents to tracker so that other peers know how to download it. The tracker will save the torrent files only, not the actual file.



Name	Date modified	Type	Size
torrents	03/12/2024 5:05 CH	File folder	

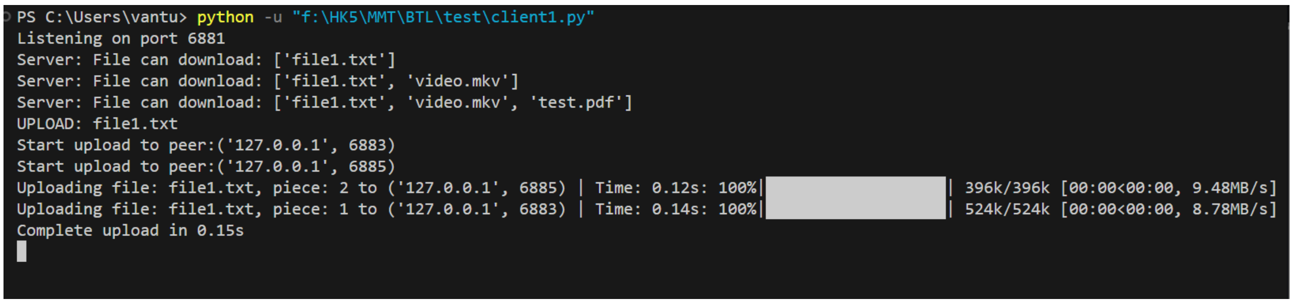


Name	Date modified	Type	Size
file1.torrent	03/12/2024 5:08 CH	TORRENT File	1 KB
test.torrent	03/12/2024 5:08 CH	TORRENT File	1 KB
video.torrent	03/12/2024 5:08 CH	TORRENT File	1 KB

Figure 4.7: Torrent folder in tracker

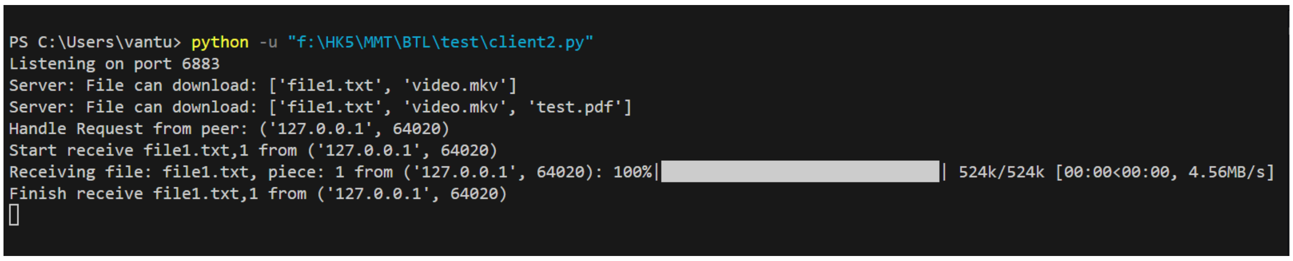
4.1.3 Uploading

Now, after specifying the available file and get the list of peers. A peer now can start upload and download files.



```
PS C:\Users\vantu> python -u "f:\HK5\MMT\BTL\test\client1.py"
Listening on port 6881
Server: File can download: ['file1.txt']
Server: File can download: ['file1.txt', 'video.mkv']
Server: File can download: ['file1.txt', 'video.mkv', 'test.pdf']
UPLOAD: file1.txt
Start upload to peer:('127.0.0.1', 6883)
Start upload to peer:('127.0.0.1', 6885)
Uploading file: file1.txt, piece: 2 to ('127.0.0.1', 6885) | Time: 0.12s: 100%|██████████| 396k/396k [00:00<00:00, 9.48MB/s]
Uploading file: file1.txt, piece: 1 to ('127.0.0.1', 6883) | Time: 0.14s: 100%|██████████| 524k/524k [00:00<00:00, 8.78MB/s]
Complete upload in 0.15s
```

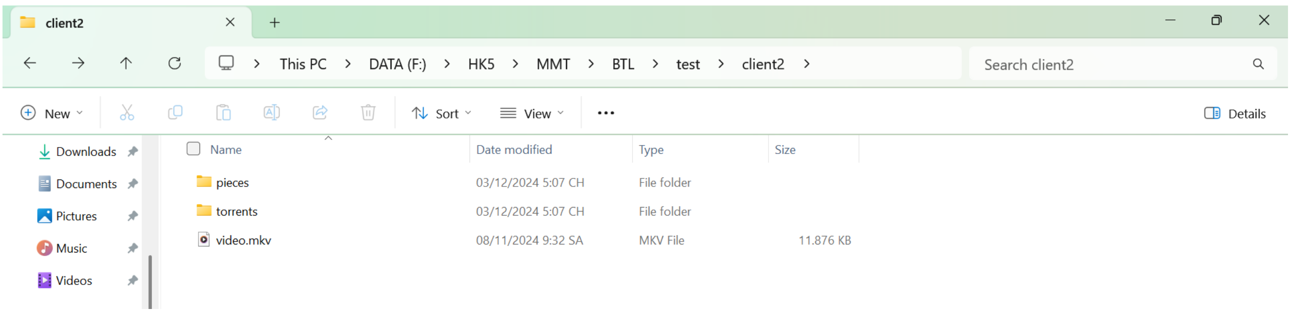
Figure 4.8: Client 1 uploads file



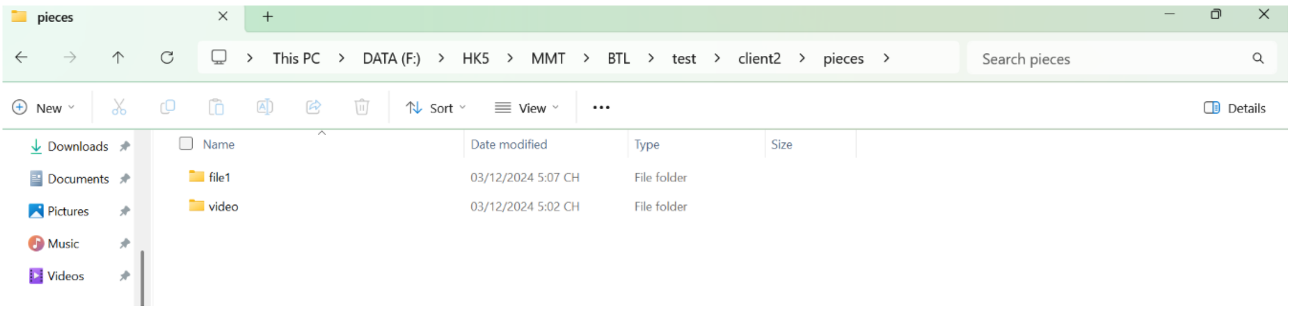
```
PS C:\Users\vantu> python -u "f:\HK5\MMT\BTL\test\client2.py"
Listening on port 6883
Server: File can download: ['file1.txt', 'video.mkv']
Server: File can download: ['file1.txt', 'video.mkv', 'test.pdf']
Handle Request from peer: ('127.0.0.1', 64020)
Start receive file1.txt,1 from ('127.0.0.1', 64020)
Receiving file: file1.txt, piece: 1 from ('127.0.0.1', 64020): 100%|██████████| 524k/524k [00:00<00:00, 4.56MB/s]
Finish receive file1.txt,1 from ('127.0.0.1', 64020)
```

Figure 4.9: Client 2 receives the file

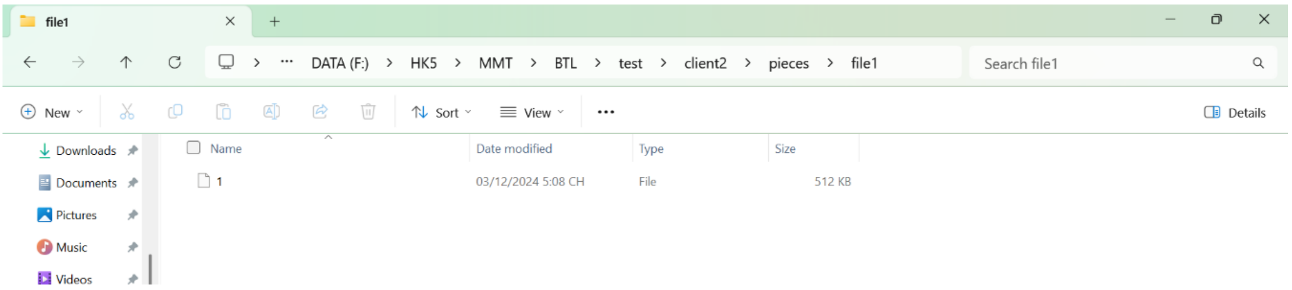
After receiving the piece from client 1, client 2 now has that piece.



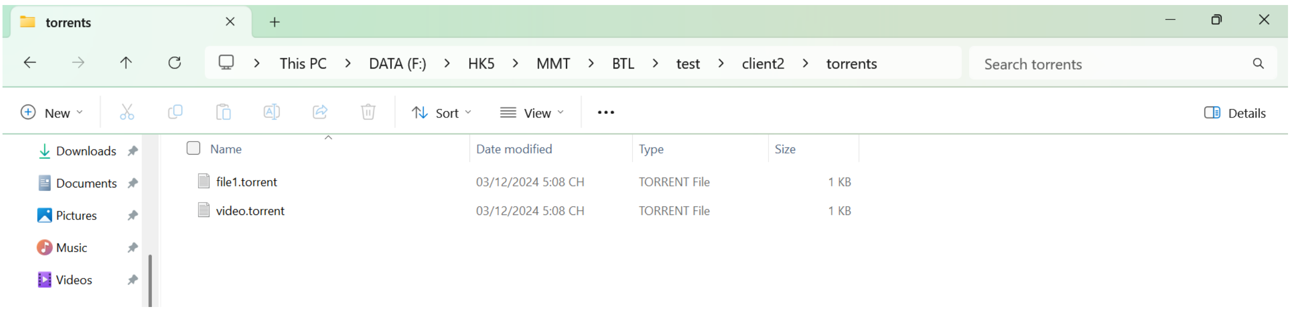
Name	Date modified	Type	Size
pieces	03/12/2024 5:07 CH	File folder	
torrents	03/12/2024 5:07 CH	File folder	
video.mkv	08/11/2024 9:32 SA	MKV File	11.876 KB



Name	Date modified	Type	Size
file1	03/12/2024 5:07 CH	File folder	
video	03/12/2024 5:02 CH	File folder	



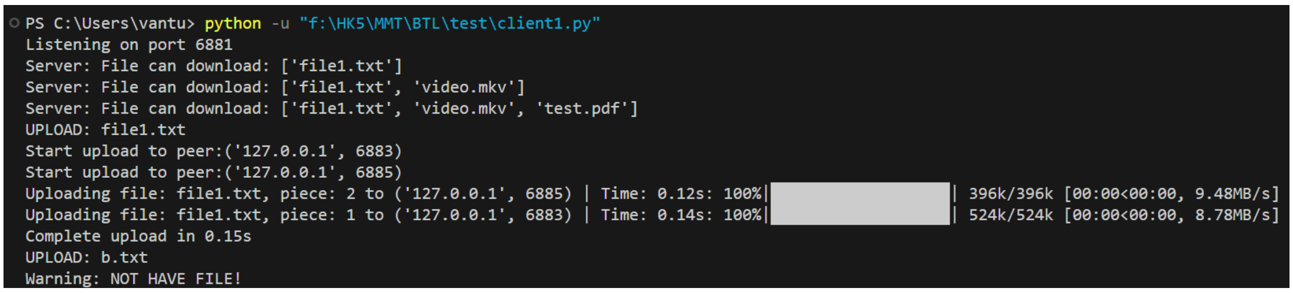
Name	Date modified	Type	Size
1	03/12/2024 5:08 CH	File	512 KB



Name	Date modified	Type	Size
file1.torrent	03/12/2024 5:08 CH	TORRENT File	1 KB
video.torrent	03/12/2024 5:08 CH	TORRENT File	1 KB

Figure 4.10: Client now has piece 1 of file1.txt

However, if a client want to upload a file that it does not have. The system will return a warning and do nothing.

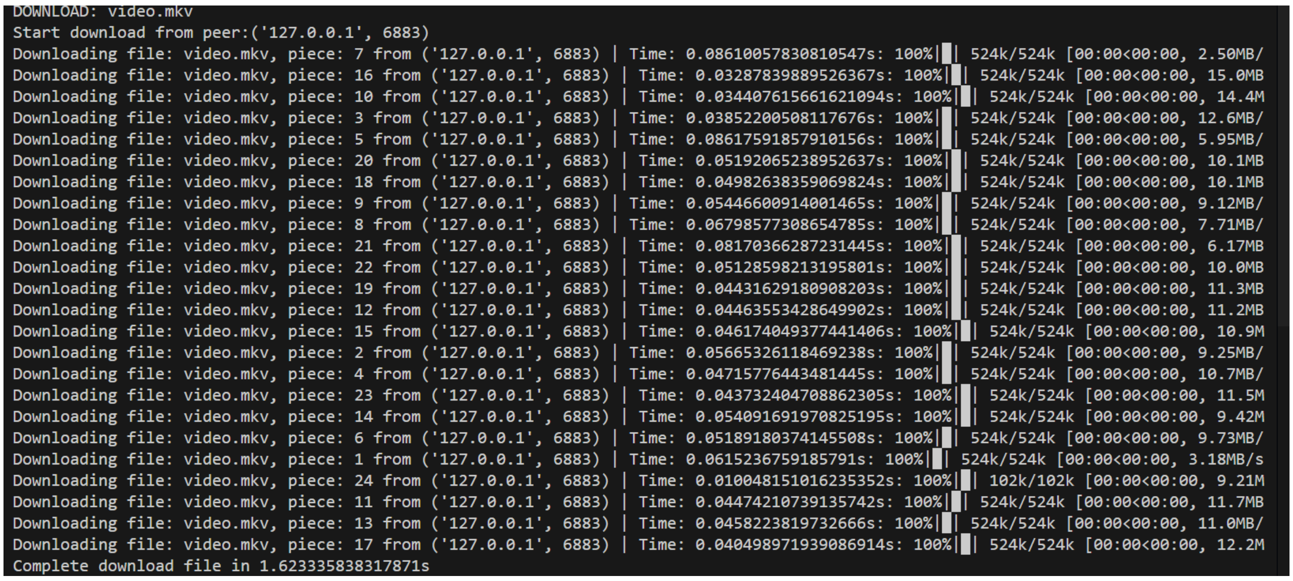


```
PS C:\Users\vantu> python -u "f:\HK5\MMT\BTL\test\client1.py"
Listening on port 6881
Server: File can download: ['file1.txt']
Server: File can download: ['file1.txt', 'video.mkv']
Server: File can download: ['file1.txt', 'video.mkv', 'test.pdf']
UPLOAD: file1.txt
Start upload to peer:('127.0.0.1', 6883)
Start upload to peer:('127.0.0.1', 6885)
Uploading file: file1.txt, piece: 2 to ('127.0.0.1', 6885) | Time: 0.12s: 100%|██████████| 396k/396k [00:00<00:00, 9.48MB/s]
Uploading file: file1.txt, piece: 1 to ('127.0.0.1', 6883) | Time: 0.14s: 100%|██████████| 524k/524k [00:00<00:00, 8.78MB/s]
Complete upload in 0.15s
UPLOAD: b.txt
Warning: NOT HAVE FILE!
```

Figure 4.11: Upload an unavailable file

4.1.4 Downloading

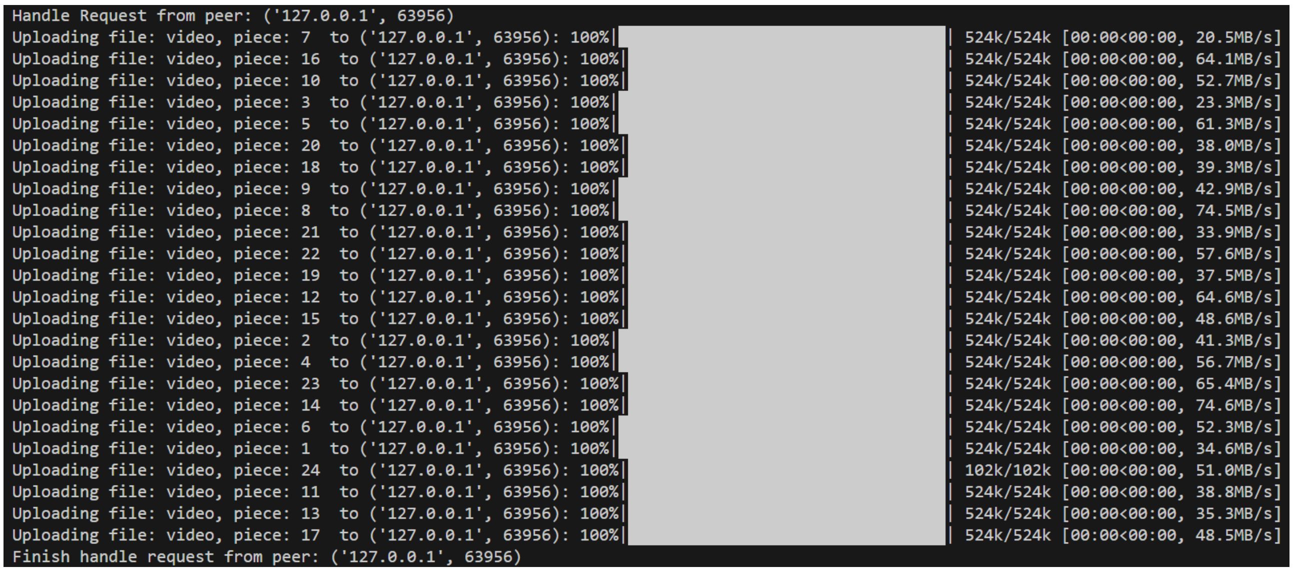
A peer can request pieces and download it from other peer. First it request a torrent, calculating the missing pieces and start to request those pieces from other peers.



```
DOWNLOAD: video.mkv
Start download from peer:('127.0.0.1', 6883)
Downloading file: video.mkv, piece: 7 from ('127.0.0.1', 6883) | Time: 0.08610057830810547s: 100%|█| 524k/524k [00:00<00:00, 2.50MB/
Downloading file: video.mkv, piece: 16 from ('127.0.0.1', 6883) | Time: 0.03287839889526367s: 100%|█| 524k/524k [00:00<00:00, 15.0MB
Downloading file: video.mkv, piece: 10 from ('127.0.0.1', 6883) | Time: 0.034407615661621094s: 100%|█| 524k/524k [00:00<00:00, 14.4M
Downloading file: video.mkv, piece: 3 from ('127.0.0.1', 6883) | Time: 0.03852200508117676s: 100%|█| 524k/524k [00:00<00:00, 12.6MB/
Downloading file: video.mkv, piece: 5 from ('127.0.0.1', 6883) | Time: 0.08617591857910156s: 100%|█| 524k/524k [00:00<00:00, 5.95MB/
Downloading file: video.mkv, piece: 20 from ('127.0.0.1', 6883) | Time: 0.05192065238952637s: 100%|█| 524k/524k [00:00<00:00, 10.1MB
Downloading file: video.mkv, piece: 18 from ('127.0.0.1', 6883) | Time: 0.04982638359069824s: 100%|█| 524k/524k [00:00<00:00, 10.1MB
Downloading file: video.mkv, piece: 9 from ('127.0.0.1', 6883) | Time: 0.05446600914001465s: 100%|█| 524k/524k [00:00<00:00, 9.12MB/
Downloading file: video.mkv, piece: 8 from ('127.0.0.1', 6883) | Time: 0.06798577308654785s: 100%|█| 524k/524k [00:00<00:00, 7.71MB/
Downloading file: video.mkv, piece: 21 from ('127.0.0.1', 6883) | Time: 0.08170366287231445s: 100%|█| 524k/524k [00:00<00:00, 6.17MB
Downloading file: video.mkv, piece: 22 from ('127.0.0.1', 6883) | Time: 0.05128598213195801s: 100%|█| 524k/524k [00:00<00:00, 10.0MB
Downloading file: video.mkv, piece: 19 from ('127.0.0.1', 6883) | Time: 0.04431629180908203s: 100%|█| 524k/524k [00:00<00:00, 11.3MB
Downloading file: video.mkv, piece: 12 from ('127.0.0.1', 6883) | Time: 0.04463553428649902s: 100%|█| 524k/524k [00:00<00:00, 11.2MB
Downloading file: video.mkv, piece: 15 from ('127.0.0.1', 6883) | Time: 0.046174049377441406s: 100%|█| 524k/524k [00:00<00:00, 10.9M
Downloading file: video.mkv, piece: 2 from ('127.0.0.1', 6883) | Time: 0.05665326118469238s: 100%|█| 524k/524k [00:00<00:00, 9.25MB/
Downloading file: video.mkv, piece: 4 from ('127.0.0.1', 6883) | Time: 0.04715776443481445s: 100%|█| 524k/524k [00:00<00:00, 10.7MB/
Downloading file: video.mkv, piece: 23 from ('127.0.0.1', 6883) | Time: 0.043732404708862305s: 100%|█| 524k/524k [00:00<00:00, 11.5M
Downloading file: video.mkv, piece: 14 from ('127.0.0.1', 6883) | Time: 0.054091691970825195s: 100%|█| 524k/524k [00:00<00:00, 9.42M
Downloading file: video.mkv, piece: 6 from ('127.0.0.1', 6883) | Time: 0.05189180374145508s: 100%|█| 524k/524k [00:00<00:00, 9.73MB/
Downloading file: video.mkv, piece: 1 from ('127.0.0.1', 6883) | Time: 0.0615236759185791s: 100%|█| 524k/524k [00:00<00:00, 3.18MB/s
Downloading file: video.mkv, piece: 24 from ('127.0.0.1', 6883) | Time: 0.010048151016235352s: 100%|█| 102k/102k [00:00<00:00, 9.21M
Downloading file: video.mkv, piece: 11 from ('127.0.0.1', 6883) | Time: 0.04474210739135742s: 100%|█| 524k/524k [00:00<00:00, 11.7MB
Downloading file: video.mkv, piece: 13 from ('127.0.0.1', 6883) | Time: 0.0458223819732666s: 100%|█| 524k/524k [00:00<00:00, 11.0MB/
Downloading file: video.mkv, piece: 17 from ('127.0.0.1', 6883) | Time: 0.040498971939086914s: 100%|█| 524k/524k [00:00<00:00, 12.2M
Complete download file in 1.623335838317871s
```

Figure 4.12: Downloading a file

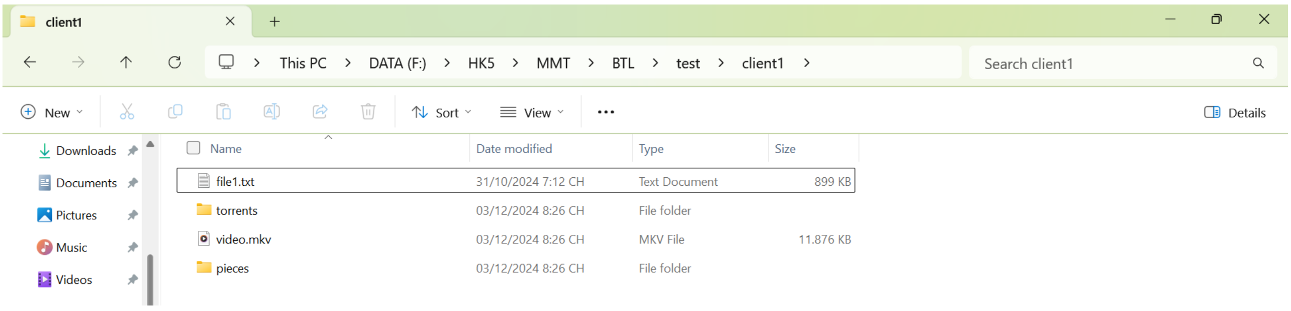
Because at the beginning, only client 2 with port number 6883 has the file video.mkv, it only request to that peer.
On the uploading side of client 2. It handles the request and upload the pieces to client 1.



```
Handle Request from peer: ('127.0.0.1', 63956)
Uploading file: video, piece: 7  to ('127.0.0.1', 63956): 100%|                    | 524k/524k [00:00<00:00, 20.5MB/s]
Uploading file: video, piece: 16  to ('127.0.0.1', 63956): 100%|                   | 524k/524k [00:00<00:00, 64.1MB/s]
Uploading file: video, piece: 10  to ('127.0.0.1', 63956): 100%|                   | 524k/524k [00:00<00:00, 52.7MB/s]
Uploading file: video, piece: 3  to ('127.0.0.1', 63956): 100%|                    | 524k/524k [00:00<00:00, 23.3MB/s]
Uploading file: video, piece: 5  to ('127.0.0.1', 63956): 100%|                    | 524k/524k [00:00<00:00, 61.3MB/s]
Uploading file: video, piece: 20  to ('127.0.0.1', 63956): 100%|                   | 524k/524k [00:00<00:00, 38.0MB/s]
Uploading file: video, piece: 18  to ('127.0.0.1', 63956): 100%|                   | 524k/524k [00:00<00:00, 39.3MB/s]
Uploading file: video, piece: 9  to ('127.0.0.1', 63956): 100%|                    | 524k/524k [00:00<00:00, 42.9MB/s]
Uploading file: video, piece: 8  to ('127.0.0.1', 63956): 100%|                    | 524k/524k [00:00<00:00, 74.5MB/s]
Uploading file: video, piece: 21  to ('127.0.0.1', 63956): 100%|                   | 524k/524k [00:00<00:00, 33.9MB/s]
Uploading file: video, piece: 22  to ('127.0.0.1', 63956): 100%|                   | 524k/524k [00:00<00:00, 57.6MB/s]
Uploading file: video, piece: 19  to ('127.0.0.1', 63956): 100%|                   | 524k/524k [00:00<00:00, 37.5MB/s]
Uploading file: video, piece: 12  to ('127.0.0.1', 63956): 100%|                   | 524k/524k [00:00<00:00, 64.6MB/s]
Uploading file: video, piece: 15  to ('127.0.0.1', 63956): 100%|                   | 524k/524k [00:00<00:00, 48.6MB/s]
Uploading file: video, piece: 2  to ('127.0.0.1', 63956): 100%|                    | 524k/524k [00:00<00:00, 41.3MB/s]
Uploading file: video, piece: 4  to ('127.0.0.1', 63956): 100%|                    | 524k/524k [00:00<00:00, 56.7MB/s]
Uploading file: video, piece: 23  to ('127.0.0.1', 63956): 100%|                   | 524k/524k [00:00<00:00, 65.4MB/s]
Uploading file: video, piece: 14  to ('127.0.0.1', 63956): 100%|                   | 524k/524k [00:00<00:00, 74.6MB/s]
Uploading file: video, piece: 6  to ('127.0.0.1', 63956): 100%|                    | 524k/524k [00:00<00:00, 52.3MB/s]
Uploading file: video, piece: 1  to ('127.0.0.1', 63956): 100%|                    | 524k/524k [00:00<00:00, 34.6MB/s]
Uploading file: video, piece: 24  to ('127.0.0.1', 63956): 100%|                   | 102k/102k [00:00<00:00, 51.0MB/s]
Uploading file: video, piece: 11  to ('127.0.0.1', 63956): 100%|                   | 524k/524k [00:00<00:00, 38.8MB/s]
Uploading file: video, piece: 13  to ('127.0.0.1', 63956): 100%|                   | 524k/524k [00:00<00:00, 35.3MB/s]
Uploading file: video, piece: 17  to ('127.0.0.1', 63956): 100%|                   | 524k/524k [00:00<00:00, 48.5MB/s]
Finish handle request from peer: ('127.0.0.1', 63956)
```

Figure 4.13: Uploading a file

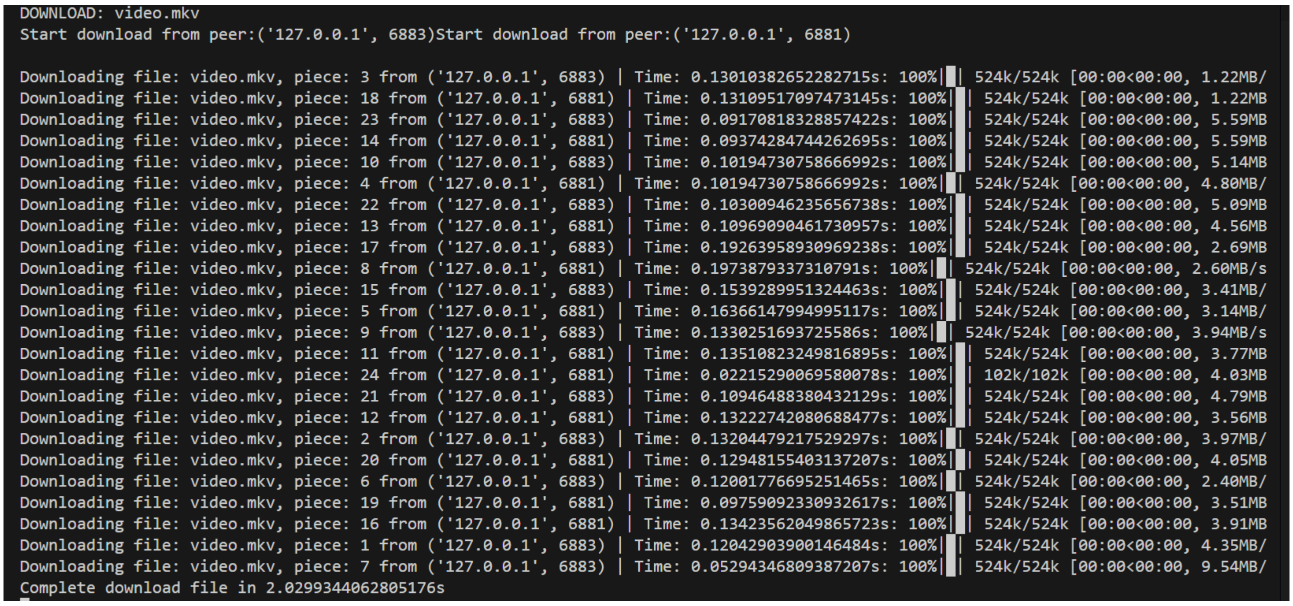
Now client 1 has the file video.mkv



Name	Date modified	Type	Size
file1.txt	31/10/2024 7:12 CH	Text Document	899 KB
torrents	03/12/2024 8:26 CH	File folder	
video.mkv	03/12/2024 8:26 CH	MKV File	11.876 KB
pieces	03/12/2024 8:26 CH	File folder	

Figure 4.14: Client 1 folder.

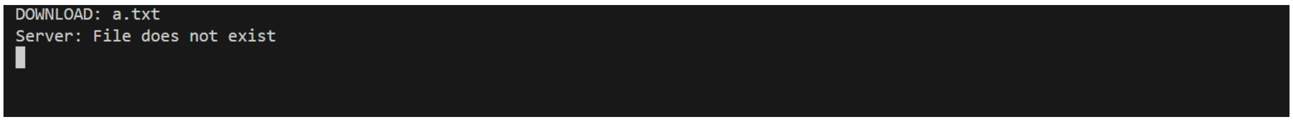
When client 3 request for the file video.mkv, now both client 1 and client 2 has that file so client 3 can request the pieces of that file from both client 1 and client 2.



```
DOWNLOAD: video.mkv
Start download from peer:('127.0.0.1', 6883)Start download from peer:('127.0.0.1', 6881)

Downloading file: video.mkv, piece: 3 from ('127.0.0.1', 6883) | Time: 0.13010382652282715s: 100%|█| 524k/524k [00:00<00:00, 1.22MB/
Downloading file: video.mkv, piece: 18 from ('127.0.0.1', 6881) | Time: 0.13109517097473145s: 100%|█| 524k/524k [00:00<00:00, 1.22MB
Downloading file: video.mkv, piece: 23 from ('127.0.0.1', 6883) | Time: 0.09170818328857422s: 100%|█| 524k/524k [00:00<00:00, 5.59MB
Downloading file: video.mkv, piece: 14 from ('127.0.0.1', 6881) | Time: 0.09374284744262695s: 100%|█| 524k/524k [00:00<00:00, 5.59MB
Downloading file: video.mkv, piece: 10 from ('127.0.0.1', 6883) | Time: 0.10194730758666992s: 100%|█| 524k/524k [00:00<00:00, 5.14MB
Downloading file: video.mkv, piece: 4 from ('127.0.0.1', 6881) | Time: 0.10194730758666992s: 100%|█| 524k/524k [00:00<00:00, 4.80MB/
Downloading file: video.mkv, piece: 22 from ('127.0.0.1', 6883) | Time: 0.10300946235656738s: 100%|█| 524k/524k [00:00<00:00, 5.09MB
Downloading file: video.mkv, piece: 13 from ('127.0.0.1', 6881) | Time: 0.10969090461730957s: 100%|█| 524k/524k [00:00<00:00, 4.56MB
Downloading file: video.mkv, piece: 17 from ('127.0.0.1', 6883) | Time: 0.19263958930969238s: 100%|█| 524k/524k [00:00<00:00, 2.69MB
Downloading file: video.mkv, piece: 8 from ('127.0.0.1', 6881) | Time: 0.1973879337310791s: 100%|█| 524k/524k [00:00<00:00, 2.60MB/s
Downloading file: video.mkv, piece: 15 from ('127.0.0.1', 6883) | Time: 0.1539289951324463s: 100%|█| 524k/524k [00:00<00:00, 3.41MB/
Downloading file: video.mkv, piece: 5 from ('127.0.0.1', 6881) | Time: 0.16366147994995117s: 100%|█| 524k/524k [00:00<00:00, 3.14MB/
Downloading file: video.mkv, piece: 9 from ('127.0.0.1', 6883) | Time: 0.1330251693725586s: 100%|█| 524k/524k [00:00<00:00, 3.94MB/s
Downloading file: video.mkv, piece: 11 from ('127.0.0.1', 6881) | Time: 0.13510823249816895s: 100%|█| 524k/524k [00:00<00:00, 3.77MB
Downloading file: video.mkv, piece: 24 from ('127.0.0.1', 6881) | Time: 0.022152900695800078s: 100%|█| 102k/102k [00:00<00:00, 4.03MB
Downloading file: video.mkv, piece: 21 from ('127.0.0.1', 6883) | Time: 0.10946488380432129s: 100%|█| 524k/524k [00:00<00:00, 4.79MB
Downloading file: video.mkv, piece: 12 from ('127.0.0.1', 6881) | Time: 0.13222742080688477s: 100%|█| 524k/524k [00:00<00:00, 3.56MB
Downloading file: video.mkv, piece: 2 from ('127.0.0.1', 6883) | Time: 0.13204479217529297s: 100%|█| 524k/524k [00:00<00:00, 3.97MB/
Downloading file: video.mkv, piece: 20 from ('127.0.0.1', 6881) | Time: 0.12948155403137207s: 100%|█| 524k/524k [00:00<00:00, 4.05MB
Downloading file: video.mkv, piece: 6 from ('127.0.0.1', 6883) | Time: 0.12001776695251465s: 100%|█| 524k/524k [00:00<00:00, 2.40MB/
Downloading file: video.mkv, piece: 19 from ('127.0.0.1', 6881) | Time: 0.09759092330932617s: 100%|█| 524k/524k [00:00<00:00, 3.51MB
Downloading file: video.mkv, piece: 16 from ('127.0.0.1', 6881) | Time: 0.13423562049865723s: 100%|█| 524k/524k [00:00<00:00, 3.91MB
Downloading file: video.mkv, piece: 1 from ('127.0.0.1', 6883) | Time: 0.12042903900146484s: 100%|█| 524k/524k [00:00<00:00, 4.35MB/
Downloading file: video.mkv, piece: 7 from ('127.0.0.1', 6883) | Time: 0.05294346809387207s: 100%|█| 524k/524k [00:00<00:00, 9.54MB/
Complete download file in 2.0299344062805176s
```

Figure 4.15: Downloading a file from multiple peers

If a peer try to request a file that is not available, the system will return a warning and do nothing.

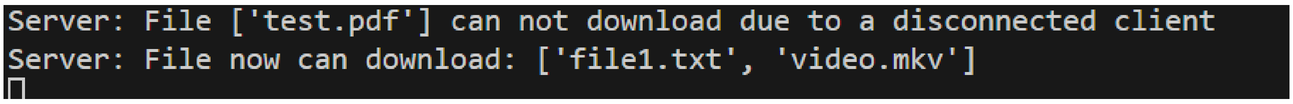


```
DOWNLOAD: a.txt
Server: File does not exist
```

Figure 4.16: Warning file not available

4.1.5 Peer disconnecting

When a peer disconnects, its files and pieces is not available anymore if no other peers has the same files and pieces. For example, client 3 is the only peer holds test.pdf, when it disconnect, that file becomes unavailble. The tracker will broadcast this information to all peers.

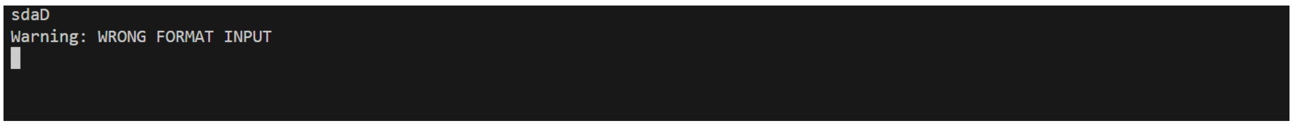


```
Server: File ['test.pdf'] can not download due to a disconnected client
Server: File now can download: ['file1.txt', 'video.mkv']
```

Figure 4.17: A client disconnect.

4.1.6 Other exception

Beside of requesting an unavailable file or uploading a file that a peer doesn't hold. The other exception is the wrong input format. The system handles it by simply throwing back a warning "WRONG FORMAT INPUT" and do nothing.



```
sdaD
Warning: WRONG FORMAT INPUT
```

Figure 4.18: Wrong format exception

5 Conclusion

In this assignment, we designed the architecture and protocol for a simple file sharing application that is similar to Bit Torrent. We implemented the application and demonstrated it in this report. Thanks for reading!