NETWORK APPLICATION REPORT

Joshua Diegaardt – DGRJOS001

Micah Retief – RTFMIC002

Nikita Martin – MRTNIK003

Table of Contents

1. Introduction
2. Application Description
3. Component Design and Implementation
    - Chunk Save
    - Tracker
    - Seeder
    - Leecher
4. Key Features and Optimizations
5. Protocol Design and Specification
    - Communication Mechanisms
    - Message Types and Structures
    - Communication Rules and States
6. Application Screenshots and Demonstration
7. Conclusions

Introduction

The assignment was concerned with developing a Peer-to-Peer file transfer application. Such a system consists of three components, namely a tracker, seeder and leecher. The tracker is aware of which peers are on the network and which files they have; the seeder is a peer that provides files for download and the leecher is a peer that requests and downloads files from a seeder. This system mimics the well-known BitTorrent application, where a peer can request files from other peers in a decentralised manner.

Our System consists of three primary components:

1. **Tracker:** Acts as the network coordinator, which maintains awareness of multiple peers and their available files in the network.
2. **Seeders:** Peers that possess complete files and makes them available for download to others.
3. **Leechers:** Peers that request and download files from seeders.

Application Description

Our application follows a hybrid Peer-to-Peer architecture, combining centralized peer discovery with decentralized data transfer. The tracker serves only as a coordinator rather than a file provider via a User Datagram Protocol (UDP) connection, keeping its bandwidth

requirements as minimal as possible, while peers communicate via a Transmission Control Protocol (TCP) connection, transferring files between each other. It consists of three python files, namely chunkSave.py, Tracker.py and p2.py.

Chunk Save

The chunk save file is responsible for splitting files into chunks, hashing these chunks and reassembling these chunks into a final file.

- Splitting into chunks: chunk save loops through all files in a given directory, gets the total number of bytes per file and then reads every 512 000 kilobytes (512KB) into a binary file. These binary files are named according to file name and chunk number, which serves as an index.
- Hashing: chunk save calculates a SHA 256 hash value for each chunk and returns it.
- Reassemble: the chunks are ordered according to index, looped through and then written to a final file. This reads in each byte per chunk in order and writes them to a final file.

Tracker

The tracker runs on a host and communicates using UDP connections. The application relies on the availability of the tracker, so this is run first. The tracker accepts UDP connections from peers (other hosts) and sends response messages to these peers when necessary. The tracker is always aware of which peers are available, as well as which file chunks they have. This information is stored, along with the IP address, port number of the TCP socket and the port number of the UDP socket of the peer.

The functionality of the tracker can be categorised according to the following:

- **Peer registration**: the tracker registers peers, including the IP address, port number of the UDP socket, port number of the TCP listening socket and file chunks that each peer has. This allows for the tracker to know which peers are available, what file chunks they offer, the UDP socket to know from where the UDP messages come, and the TCP listening socket for other peers to initiate TCP connections when downloading files. After a peer registers with the tracker it gets marked as ‘active.’
- **File request**: the tracker receives a message from a peer, which would be a leecher in this case, which contains the name of the requested file. The tracker takes the name of this file, searches through the dictionary of available file chunks, and returns a list of the IP address and TCP listening socket of the peer that has these chunks. In addition, it sends the SHA 256 hash values of each chunk to the leecher.
- **Reseed**: the tracker receives a message from a peer with an update of the available chunks that it has. This is done when a peer downloads a chunk from another peer, and is also done when a peer wants to update the chunks that it offers. This ensures that the tracker has the most recent list of available chunks, and which peers have them.
- **List files**: the tracker receives a request for a list of all available files from a peer and responds by sending this list to the peer. This helps the peer know which files it can download.
- **Heartbeat**: the tracker receives a ‘heartbeat’, or a ping, from the peer every minute to communicate that the peer is still available. The tracker checks for a heartbeat every

minute, and if there are no heartbeats from a peer after 3 minutes then the peer gets marked as ‘away.’ If there are no heartbeats after 5 minutes then the peer, as well as the file chunks that it offers, are removed from the list of available peers.

- **Exit**: the tracker removes a peer as well as the file chunks that it offers. This is done when a peer willingly leaves the application or after it is removed after being inactive.

Seeder

The seeder provides file chunks for other peers to download. The seeder communicates with the tracker via UDP messages and sends files to requesting peers (leechers) via a TCP connection. The seeder’s functionality can be categorised into the following:

- **Registration**: the peer registers on the application by sending a UDP message to the tracker, becoming a seeder immediately afterwards. This process begins by creating a UDP socket for communicating with the tracker and a TCP listening socket. Both sockets are run by individual threads that are used to communicate with the tracker and other peers. After this, a directory that the user would like to make available to other peers is selected; any files in the chosen directory will be split into 512kb binary files, stored in a directory called “chunks” and the metadata of the file of is returned. These chunks are named according to the file name and chunk number, which serves as an index. The metadata consists of the file name, number of bytes in the file, number of 512KB chunks that the file can be split into and the SHA256 hash values for each chunk of the file. After the files have been split into chunks and the metadata has been retrieved, the seeder then sends the UDP socket information, TCP listening socket information and file metadata to the tracker.
- **Reseed**: the seeder sends an updated list of files, or reseeds, to the tracker. The reseed function allows the seeder to choose a new directory to make available to other peers. This is done by clearing the “chunks” directory and splitting the files of the new chosen directory into chunks and storing them in the “chunks” directory. This replaces the old chunks with new chunks.
- **Update availability**: the seeder notifies the tracker of any file changes made to the current available directory such as adding or deleting files. The implementation is similar to the implementation used during reseeding.
- **List files**: the seeder requests a list of the available files for download from the tracker.
- **Exit**: the seeder exits the application by sending a message to the tracker requesting for it to be removed. The tracker then removes the information of the seeder along with the file chunks that it offers. On the seeder side, all files are removed from the “chunks” directory to ensure that when a peer registers to be a seeder, there are no remnants of the previous connection.

Leecher

The leecher is a peer that requests and downloads a file from another peer, which would be a seeder in this case. In addition to having the functionality of a seeder, the functionality of a leecher can be categorised into the following:

- **Request a file**: the leecher requests a file by sending a message to the tracker which contains the file name. The tracker searches through the available files and the seeder of each file, returning the information of all seeders, the number of 512KB

chunks that make up the file and the hash values for each chunk. The tracker also returns a download plan for parallel downloads, where it displays which chunk will be downloaded from which seeder. If there are no seeders for the file then the tracker communicates that there are no seeders for the requested file.

- **Download a file**: after the seeder information is returned, the leecher initiates a TCP connection with the seeder by connecting to the TCP listening socket of the seeder. This then creates a separate TCP socket for the transfer of the requested file. Thereafter, the leecher downloads each chunk from the seeder, calculates and compares their hash values until all chunks are received. It then places each chunk into a directory called “downloads.” With each chunk download, the leecher tries to download a given chunk 3 times; if unsuccessful, the leecher notes that the chunk is missing and requests these missing chunks from available seeders after the download. After a chunk is downloaded, the tracker is notified that the leecher has received a specific chunk from the requested file; the leecher also keeps track of which chunks have been downloaded.
- **Reassemble a file**: after all chunks are downloaded, they are ordered according to chunk to index and then reassembled into the entire file.

Features

- **Parallel Downloads**:
  Leeches can request chunks from multiple Seeders simultaneously, while the Tracker provides a download plan to balance load between the peers. This improves the download speed compared to single peer download.
- **Automatic Re-Seeding** – Lechers automatically transitions to seeders after complete downloads. This increases the file availability in the network, and it ensures proper distribution of rare chunks.
- **Heartbeat Mechanism** – This prevents inactive peers from cluttering the tracker. It automatically removes the peers if they are away for more than 5 minutes, and it automatically removes failed or disconnected peers.
- **Chunk-based File Transfer** – Files are split into fixed sized chunks (512 KB) for fast and efficient file distribution. It handles interrupted downloads efficiently and also facilitates parallel transfers from multiple peers.
- **Failure Recovery** – Implements a retry mechanism for failed chunk downloads, and it attempts to request missing chunks from alternative peers if they have the file. It also handles peer disconnections gracefully.
- **File chunk verification**—When a file is split into 512KB chunks, a hash value is calculated using SHA 256 and sent to the tracker. When a leecher requests a file chunk, the tracker sends this hash value to the leecher along with the information of the seeder. After a leecher initiates a connection and downloads the file chunk, it calculates the hash value and compares it to the value sent by the tracker. If the hash value of the received chunk matches the value sent by the seeder, the file chunk is kept and saved in the “downloads” directory. After all files are received, the leecher reassembles the file.

Protocol Design and Specification

1.1 Communication Mechanisms

UDP (Tracker Communication & Peer Discovery): This is used for lightweight, fast communication between the peers and the tracker. This minimizes the overhead for operations such as peer registration, heartbeats and file requests.
TCP (File Transfer): This ensures reliable and ordered data transfer between peers. TCP are connection-oriented and ensures that each chunk is transferred completely and in order.

1.2 Message Types

The Protocol consists of three types of messages:

Command messages:

The purpose of command messages is to initiate or to terminate communication between parties in the network. These JSON-formatted messages include:

- 'REGISTER' - Sent by peers to the tracker, for when a peer registers in the network and specifies the file chunks that they have available to share.
  Format: "**type**: REGISTER, **peer_udp_address**: {udp_socket_address}, peer_tcp_address: {tcp_socket_address}, **files**: [file_name1: chunk_list, file_name2: chunk_list, …], **metadata**: [file_name1: file_matadata, file_name2: file_metadata]"
- 'REQUEST' – Sent by a peer/leecher to the tracker when they would like to request a file.
  Format: "**type**: REQUEST, **peer_upd_address**: {peer_udp_address}, **file_request**: {file_name}"
- 'RESEED' – Sent by peers to notify the tracker that they now have additional files or that they want to share a new file.
  Format: "**type**: RESEED, **peer_udp_address:** {udp_socket_address}, **filename**: {file_name}, **chunk**{chunk_num_of_file}"
- 'HEARTBEAT' – Sent periodically by peers to inform the tracker that they are still active.
  Format: "**type**: HEARTBEAT, **peer_address:** udp_socket_address"
- 'EXIT' – Sent by a peer to the tracker when the peer would like to exit the network, so that the tracker can remove it from the database.
  Format: "**type**: EXIT, **peer_udp_address:** {udp_socket_address}, **peer_tcp_address:** {peer_tcp_address}"
- LIST_FILES: Requests available files from the tracker.
  Format: "**type**: LIST_FILES, **peer_udp_address**: {udp_socket_address}"

Data transfer messages:

Data transfer messages are used to carry the data that is exchanged between leechers and seeders in the network. Data transfer messages include:

- 'CHUNK_REQUEST' – Sent by leechers to seeders, requesting a specific file chunk, after the TCP connection is established between the two.
  Format: "**type**: CHUNK_REQUEST, **filename**: {file_name}, **chunk_num**: {chunk_num}"
- 'REQUEST_CHUNKS' – This is sent by the peer to the tracker during file transfer when there are chunks missing.
  Format: "**type**: 'REQUEST_CHUNKS', **peer_udp_address**: {udp_socket_address}, **filename**: {filename}, **chunks**: {chunks_requested)"

Control_messages:

The purpose of control messages is to manage the communication between peers in the network, such as message acknowledgment or error messages.

- ‘ACK’: Acknowledges the receipt of a message, from the tracker, or when the peer downloaded chunks from another peer.
- ‘ERROR’: Indicates an error (e.g requested chunk not found)
  Format: “error: {error messages}

1.3 Message Structure

The message consists of header and a body:

Header Fields (included in most message fields) :

- type (string):- Identifies the type of message – all in uppercase, eg. ‘REQUEST’
- peer_udp_address(tuple):- identifies the UDP socket address of the peer – which is used when connecting to the tracker. It contains the peer IP address and the peer UDP port number.
- peer_tcp_address(tuple):-identifies the TCP socket address of the peer – which is when connecting to another peer for file request. It contain the peer IP address and peer TCP port number
- **Body fields** – (present in specific message fields, such as the file information that is requested or sent):
  - filename (string):- the name of the requested or transferred file,
  - file_chunks:- the dictionary of the files and their chunks requested or transferred between the peer and tracker.
  - metadata:- a dictionary containing the metadata of each file in the system
  - chunk_num:- the chunk number to be sent/requested
  - data (bytes):- the actual chunks of data for file transfer

Example REGISTER Message (Peer – Tracker)

Format: {

“**type**: REGISTER,

“**peer_udp_address”**: ["192.168.1.100", 5000],

“**peer_tcp_address”**: ["192.168.1.100", 6000],

“**files”**: { "file1.mp4": [0, 1, 2, 3], “SeedStrorm.jpg”: [0,1,2] },

“**metadata”**: { "file1.mp4": { "size": 1048576, "num_chunks": 4 } }

}

Communication Rules and Peer States

Peer States:-

- Active – The peer has successfully registered with the Tracker, and is sending heartbeats regularly to the tracker.
- Away – The peer is neither engaging with the tracker or any other peer, nor sending heartbeats to the tracker within a period of 120 seconds.
- Removed – The peer is deleted from the tracker after 300 seconds of inactivity.

Transition Rules:

Initial Registration – Peers register with the tracker.

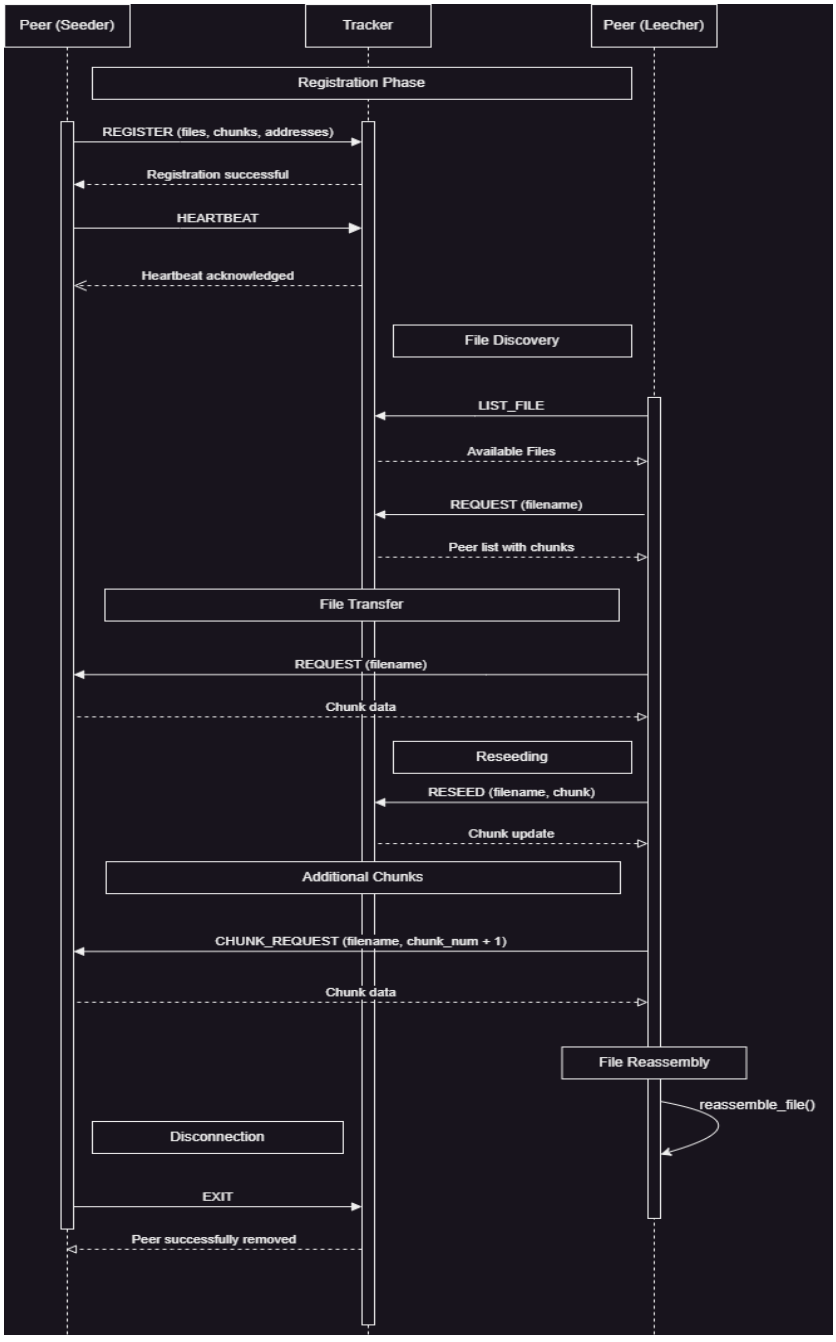
Active – A peer transitions to Active once they start sending HEARTBEATS to the tracker.

Away – A peer does not send a HEARTBEAT within 120 seconds; it is marked as away.

Removed – No activity for 300 seconds (5 minutes)

Sequence Diagram

Figure 1: Sequence Diagram Showing P2P Message Protocols



Testing and Performance:

The system was tested under various conditions to ensure that the system operation is robust.

Test Scenarios:

1. Multiple simultaneous downloads: The system performance with concurrent downloads was tested. This is managed by multiple threads, accepting connections from multiple peers simultaneously when downloading files, as well as sending files at the same time.
2. Peer failure during file transfer: Tested to ensure graceful recovery when peers connect. When peers disconnect during file transfer, the system has at most 3 retries to check if another peer has the file chunk and attempts to leech from them. If unsuccessful, the system prints a message that downloading the file was unsuccessful.
3. Network congestion simulation: Testing system behaviour under limited bandwidth. We use UDP connection with the tracker, which is lightweight and minimizes overhead on operations, and with TCP file transfer, we split the sending of file chunks between multiple peers, to minimize the load of one peer when sending chunks. The TCP connection also only allows at max 5 peers to connect.
4. Large file transfers: Verifying correct handling of files spanning many chunks.

Observed Behaviours

- Parallel Downloads – provide up to 3.5x speed improvement over sequential downloads
- Heartbeat mechanisms – which successfully detected and removed inactive peers
- Chunk distribution – remained balanced with frequent peer download

Application Screenshots

Registration: Before a peer can interact with the system, they first must specify the folder containing files that they would like to share with the system:

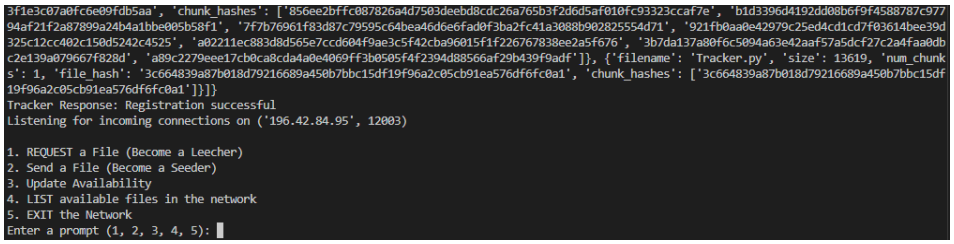
```
********************SEEDSTORM BitTorrent System********************
-Welcome to the SEEDSTORM BitTorrent System.
-A high-speed, peer-to-peer file-sharing network where peers connect, share, and storm data across the network.
-Tracker Connection: We use UDP to track and connect peers.
-File Transfer: Peers communicate via TCP for reliable data exchange
-Seeders keep the network alive
-Leechers get the files they need.

Enter the folder to share: format = ./folderName
./
```

When a peer specifies the folder with the files that they would like to share, each file within the file is split into chunks, and these file chunks gets stores into a ‘./chunks’ directory:

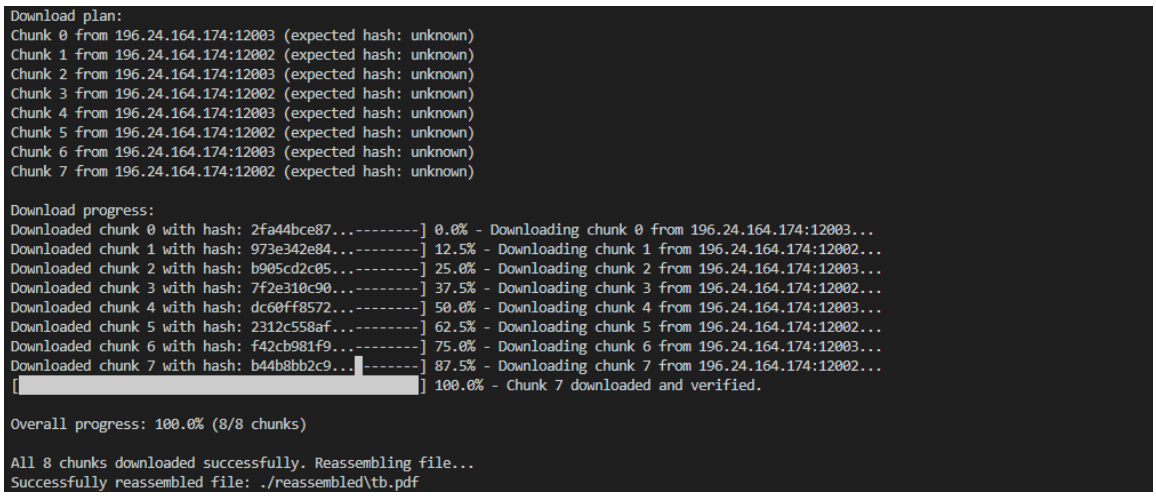
```
File chunkSave.py split into 1 chunks with hashes.
Chunk hash information saved to ./chunks\chunkSave.py_hashes.json
Skipping directory: ./downloads
Processing file: GUI.py (Size: 30949 bytes)
Saved 30949 bytes to ./chunks\GUI.py_chunk_0.bin with hash 9d947c0cf1...
File GUI.py split into 1 chunks with hashes.
Chunk hash information saved to ./chunks\GUI.py_hashes.json
Processing file: lorem.html (Size: 1166029 bytes)
Saved 512000 bytes to ./chunks\lorem.html_chunk_0.bin with hash 8311c45426...
Saved 512000 bytes to ./chunks\lorem.html_chunk_1.bin with hash ed327e291c...
Saved 142029 bytes to ./chunks\lorem.html_chunk_2.bin with hash 2889fe0a78...
```

For file verification purposes, the file data is sent to the tracker containing the filename, size, the number of chunks per file, and the hash value for each chunk. It registers with the tracker, making the peer an automatic seeder.

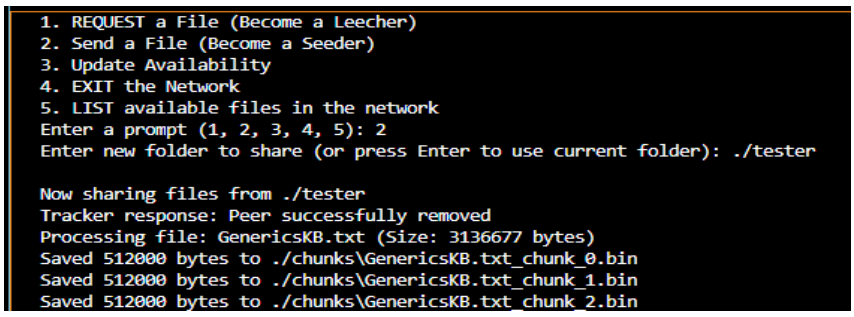
Prompts: 1 – Request a file (Leech), 2 – Send a file (Seed/Reseed), 3 – update tracker, 4 – List available files, 5 – Exit System



1: Request a File:- When a leecher requests a file, the tracker gets all the seeders with the file chunks for that file, and then the leecher makes connections with those seeders to send the file. Below shows the download plan and download progress. Once all the file chunks for the file have been received, the leecher then reassembles the files, and then reseeds the file –updating the tracker.



2: Reseeding: Allows the peer to send a new, or updated directory, to be added to the tracker database.



3: Update the tracker

```
Saved 9681 bytes to ./chunks\Tracker1.py_chunk_0.bin with hash 3e8168fab4...
File Tracker1.py split into 1 chunks with hashes.
Chunk hash information saved to ./chunks\Tracker1.py_hashes.json
Tracker Response: Registration successful
Updated file availability. Currently sharing 12 files.

1. REQUEST a File (Become a Leecher)
2. Send a File (Become a Seeder)
3. Update Availability
4. LIST available files in the network
5. EXIT the Network
Enter a prompt (1, 2, 3, 4, 5):
```

4: List the available files in the system

```
1. REQUEST a File (Become a Leecher)
2. Send a File (Become a Seeder)
3. Update Availability
4. LIST available files in the network
5. EXIT the Network
Enter a prompt (1, 2, 3, 4, 5): 4
('chunkSave.py', 'GUI.py', 'lorem.html', 'Newton Practice.pdf', 'No one should have this.txt', 'p2.py', 'Peer.py', 'Tracker.py')
```

5: Exit the system

```
1. REQUEST a File (Become a Leecher)
2. Send a File (Become a Seeder)
3. Update Availability
4. LIST available files in the network
5. EXIT the Network
Enter a prompt (1, 2, 3, 4, 5): 5
Shutting down...
Tracker Response: Peer successfully removed
```