CSC3002F Networks Assignment 1 Report

Team 44
INDSHR002 - Shriya Inderpal
JWNAR001 - Aria Jeewon
KSHKAV001 - Kavya Kaushik

a) <u>Code Design and Functionality</u>

- The system implement a Mini Peer-to-Peer File Sharing system, the three components are as follows:

1) **tracker.py:**
    - The file keeps track of all seeders that are available on the network and listen for periodic updates to ensure their availability. It implements a UDP connection to listen to incoming messages. Since UDP does not implement secure data transfer, it is unreliable for data transfer rather it is much more suitable for tracking peer to peer communication in a mini BitTorrent file sharing system.
    - It implements a UDP server that listen for incoming seeder registration requests which are then registered as available seeders. A seeder registers with its IP address, port number and file name that it holds.
    - The tracker implements a UDP server that through which it shares the list of seeders available for a certain file that is requested by the leecher.
    - The tracker implements a UDP server to process a seeders period availability update. When an update has not been received for 10 seconds the tracker removes the seeder as it is considered in inactive.
2) **seeder.py:**
    - The file runs a seeder instance by taking a port number as an argument.
    - The seeders responsibility is to ensure it registers itself to the tracker by sending a UDP message *REGISTER {file_name} {number_of_chunks} {seeder_port_number}* and send periodic updates to the tracker to maintain its availability and ensure that it remains listed as an active peer.
    - When a leecher requests for a file data using GET_CHUNKS on a TCP server, the seeder is notified, and the connection is used for secure file transfer. The seeder then sends the file in fixed- sized chunks of 512kb to the leecher along with a SHA256 hash code to verify the file integrity. To allow for multiple leecher downloads the tracker implements multi-threading to allow multiple leechers to download from multiple seeders parallelly.
    - Both the seeder and leecher use TCP client/server programming as it is a reliable protocol and ensures that data is delivered accurately and in the correct order.
    - The seeder periodically notifies the tracker that it is available every 5 seconds to help tracker track available seeders.
3) **leecher.py:**

- The leecher makes a request (GET_SEEDERS) to the tracker to get a list of all the available seeders which have the file the leecher wants to download. The leecher then receives a response from the tracker with the list of available seeders and the chunks in each file.

- It then evenly distributes the number of chunks based on the number of leechers while ensuring all chunks (fixed sized 512kb) are allocated across the seeders. The leecher then implements *Parallel Downloads* by starting multiple threads which request the specific chunks from each of the seeders. The chunks are then combined to reconstruct the original file. Error checking functionality is implemented to ensure that all chunks are received before the file is reconstructed and if any chunks are missing the user is notified that the file download is incomplete.
- The leecher saves the new file in a folder (*seeder_<portnumber>*) making it available to access it once it runs as a seeder.
- Once the leecher has successfully downloaded the file, it registers itself to the tracker as a *seeder.* This happens by running a subprocess of seeder.py with the relevant port number. The *seeder* can now share its file chunks to other leechers over a TCP connection. Error handling is implemented so that leecher does not become a seeder until it has the complete file.
- In the leecher.py, there are various error handling functions used to ensure that messages are being sent through the components correctly. The file starts of by running 4 seeders subprocesses in the background which play the role of the initial seeders when the network is started. It also runs another seeder subprocess when it converts to a leecher. Extensive error handling is implemented to ensure that each process is tracked in an array and terminated when the user exits the program or if any unexpected error crashes the program. This is critical to ensure there are no subprocess running in the background without proper termination.

b) Additional Features

We have included 4 additional features.

a. **Parallel Downloads:**

- This allows leechers to download different chunks of the same file from multiple seeders simultaneously, which in turn reduces the download time of a file.
- The leecher sends a request message to the tracker who then responds with a list of active seeders containing that specific file.
- The leecher splits the chunks evenly between the seeders and request specific chunks from each seeder.
- The seeders that contain that file split it into uniform size chunks.
- The leecher then downloads the different chunks from multiple seeders at the same time, and then reassembles them

b. **File Integrity Verification:**

- The leecher and seeder make use of SHA-256 hashing to ensure that the downloaded file is in the correct order, complete and uncorrupted.
- Before sharing the file, the seeder calculates a SHA-256 hash code for each chunk of the file and shares this with the tracker along with the other chunk data.
- When the leecher downloads the file, it calculates the hash for each chunk and compares it with the hash code provided by the seeder.
- If the hash codes match, the file is reassembled by the leecher and a verification message to printed. If not, the leecher skips those chunks of the file, and results in incomplete file download which is correctly handled.

c. **Re-Seeding:**

- Re-seeding allows for leecher to convert to seeders once they have downloaded a file and allows other leechers to download from them.
- Once the leecher downloads a file, it becomes a seeder by sending a registration message with its port number, IP address and file information to the tracker.
- After the leecher turned seeder has registered with the tracker, the tracker then adds it to the list of available seeders containing the file and this then enables other leechers to download from it.
- This allows for files to remain accessible even if the original seeder has gone offline.

d. **GUI:**

  - The graphical user interface used in the Peer- to-peer file sharing application is a progress bar that is used to indicate the downloading process from the seeder.
  - The progress bar tracks the number of chunks downloaded of the total number of chunks of the file.
  - In the tracker.py, there is a simple GUI that prints the list of available seeders, including their IP Adrees, Port Number, number of chunks is includes and the time it was last active. The GUI regularly updates as new seeders are added/removed from available seeders.

c) Protocol Messages: Structure and Sequence Diagrams

Each protocol consists of a header and a body. The header of the message is used to identify the type of message and provide information about the available data (REGISTER, REQUEST, GET_CHUNKS, AVAILABLE) and the body of the message provides the actual data that is sent over the connection.

Protocol Message 1: GET_SEEDERS {FILE_NAME}

Header:

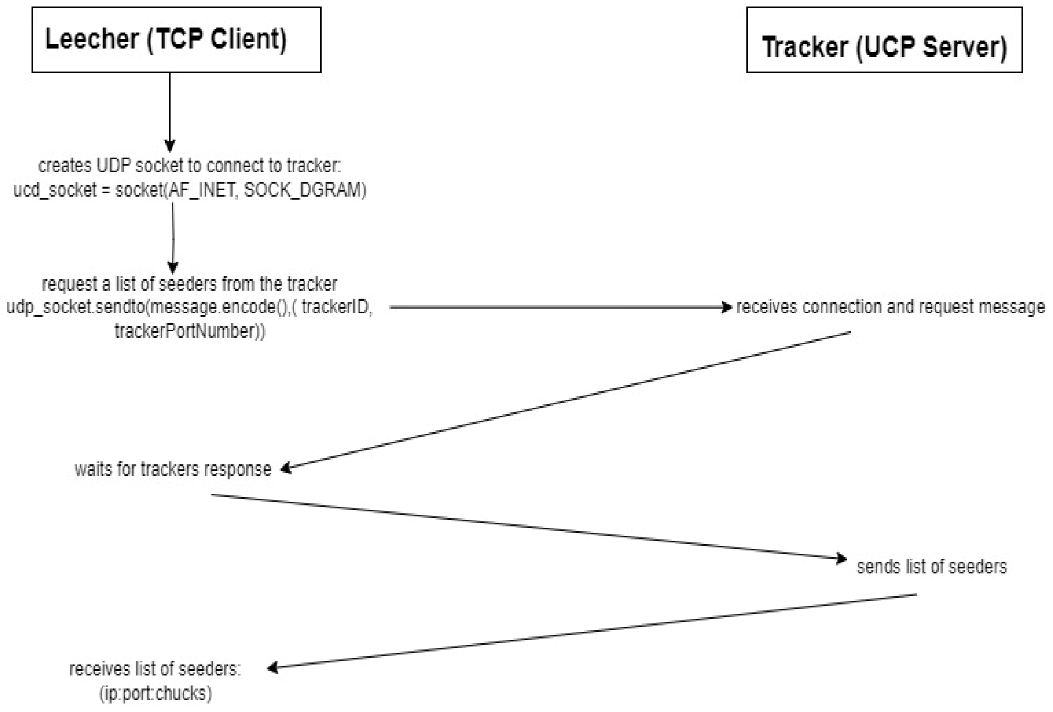
- GET_SEEDERS – message type

Body:

- {FILE_NAME} – name of the file that is requested

It's a **Command** type message. This holds the communication between the leecher and tracker as the leecher requests a list of seeders from the tracker.

Sequence diagram from Protocol message 1:


Leecher (TCP Client)
Tracker (UCP Server)
creates UDP socket to connect to tracker:
ucd_socket = socket(AF_INET, SOCK_DGRAM)
request a list of seeders from the tracker
udp_socket.sendto(message.encode(),(trackerID, trackerPortNumber))
receives connection and request message
waits for trackers response
sends list of seeders
receives list of seeders: (ip:port:chucks)

Protocol Message 2: GET_CHUNKS {file_name} {start_chunk} {end_chunk}

Header:

- GET_CHUNKS – message type

Body: contains information regarding the requested file

- {file_name} - the name of the file requested by the leecher
- {start_chuck} - chunk at which to start the download from for each seeder
- {end_chunk} - chunk at which to stop the download for each seeder

It's a **Command** type message. This communication is between the leecher and the seeder. Once the leechers receives the list of seeders, it requests for the chunks from the seeder all via a TCP connection.

Protocol Message 3: {chunk length} {chunk_data} {chunk hash}

Header:

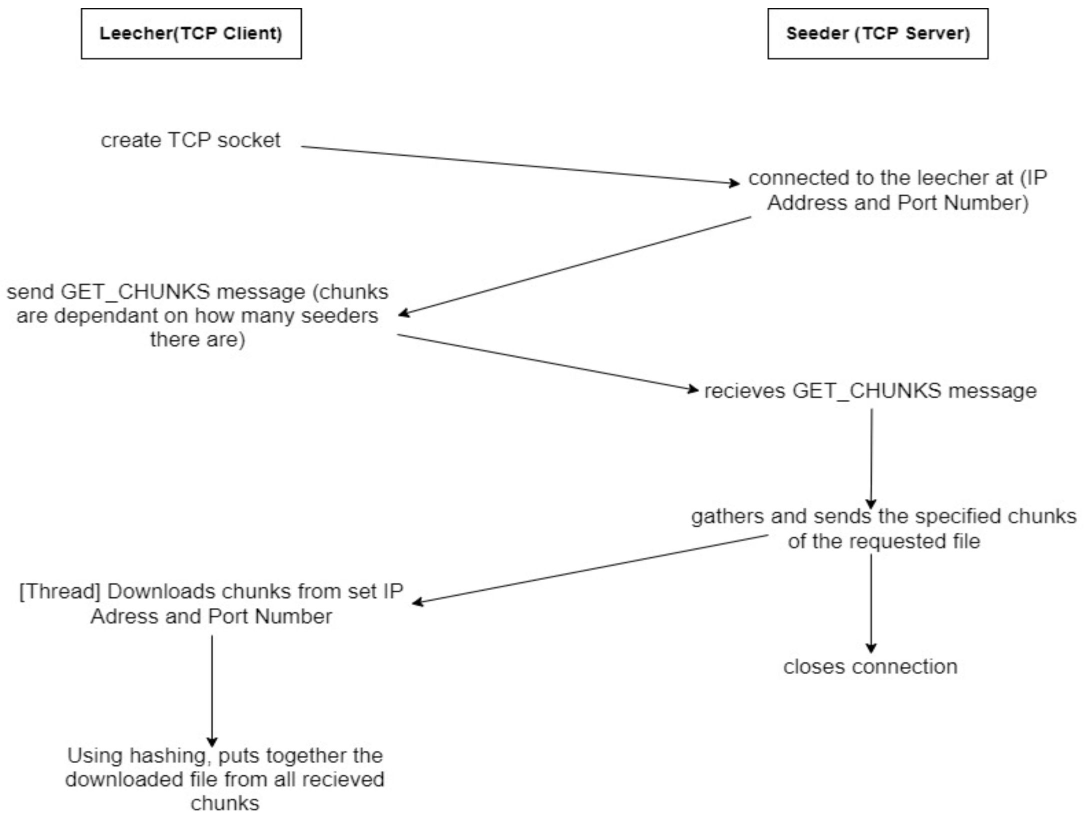
- {chunk length} – gives information of chunk size

Body: data is sent with the protocol

- {chunk data} – the file data that is sent
- {chunk hash} – for chunk verification

It's a **Data Transfer** type message. This is used by the seeder when leecher requests chunks.

Sequence diagram from Protocol message 2 and 3:


Leecher(TCP Client)
Seeder (TCP Server)
create TCP socket
connected to the leecher at (IP Address and Port Number)
send GET_CHUNKS message (chunks are dependant on how many seeders there are)
recieves GET_CHUNKS message
gathers and sends the specified chunks of the requested file
[Thread] Downloads chunks from set IP Adress and Port Number
closes connection
Using hashing, puts together the downloaded file from all recieved chunks

<u>Protocol Message 4</u>: REGISTER {file_name} {file_chunks} {seeder_port}

Header:

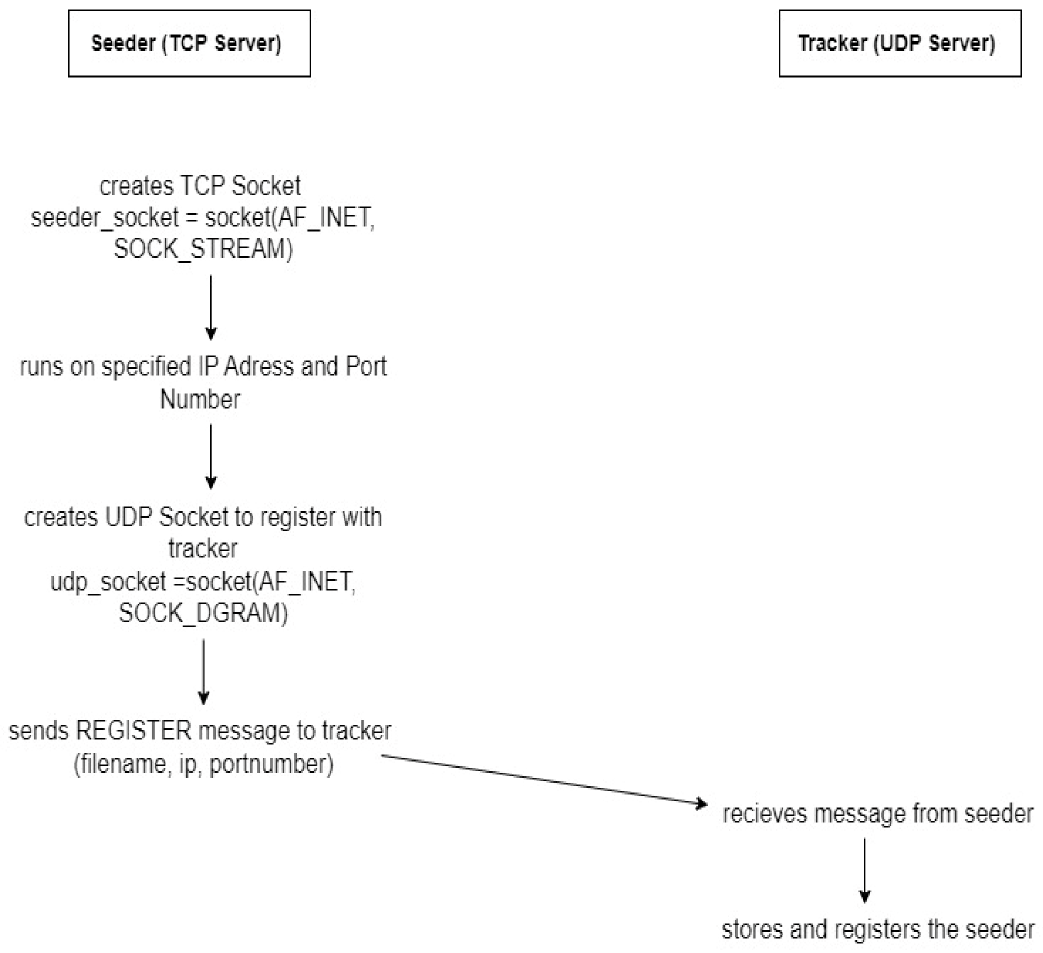
- REGISTER– message type

Body: contains the data that is sent with the protocl

- {file_name} - the name of the file that the seeder is being registered for
- {file_chunks} - the number of chunks that the file is broken into
- {seeder_port} - port number of the seeder that forms the connection with the tracker

It's a **Command** type message. This holds the communication between the seeder and tracker to register itself in the tracker. The Sequence diagram below is also used for when the leecher becomes a seeder and registers itself with the tracker.

Sequence diagram from Protocol message 4:


Seeder (TCP Server)
Tracker (UDP Server)
creates TCP Socket seeder_socket = socket(AF_INET, SOCK_STREAM)
runs on specified IP Adress and Port Number
creates UDP Socket to register with tracker udp_socket =socket(AF_INET, SOCK_DGRAM)
sends REGISTER message to tracker (filename, ip, portnumber)
recieves message from seeder
stores and registers the seeder

<u>Protocol Message 5</u>: AVAILABLE {SEEDER_IP}:{SEEDER_PORT}

Header:

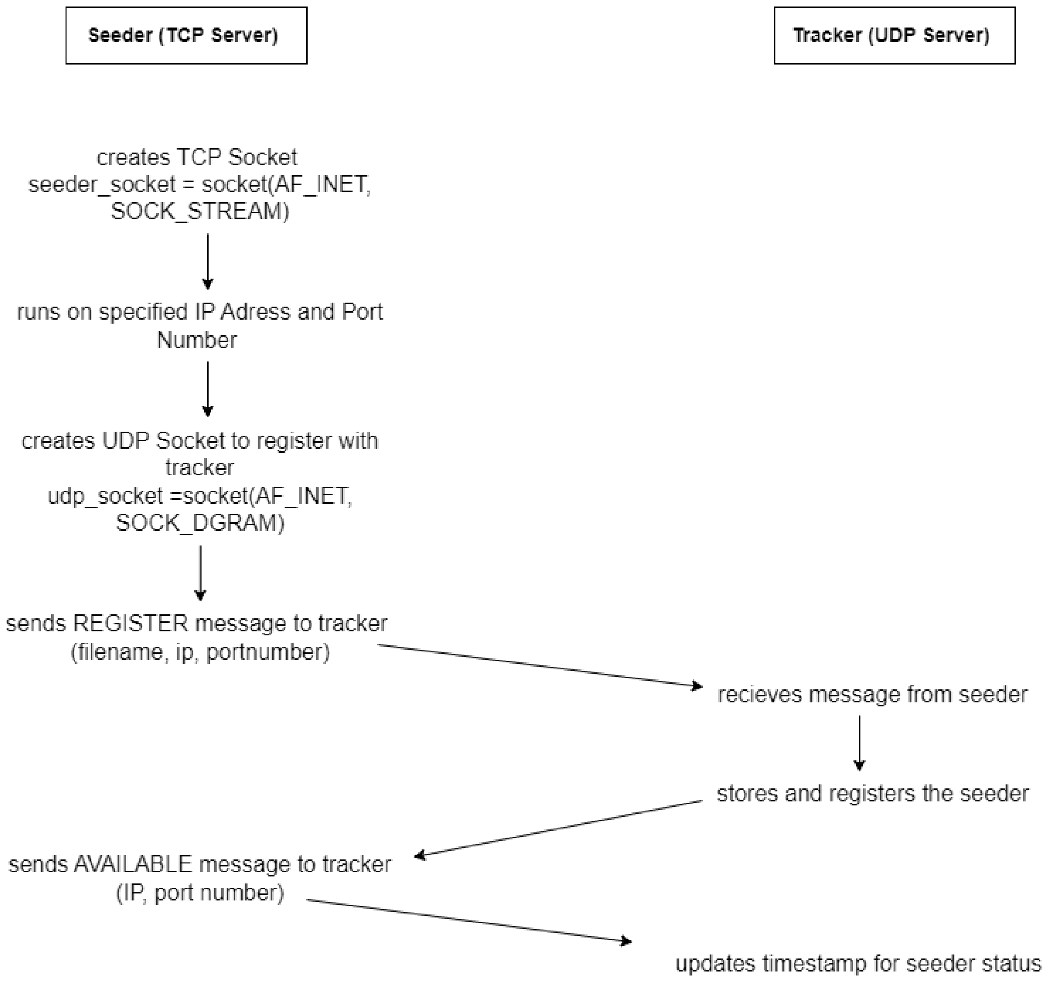
- AVAILABLE – message type

Body: information regarding the available seeders

- {SEEDER_IP} - IP address of the available seeder
- {SEEDER_PORT} - port number of available seeder

It's a **Command** type message. This is used to inform the tracker that a seeder is available to use.

Sequence diagram from Protocol message 5:


Seeder (TCP Server)
Tracker (UDP Server)
creates TCP Socket seeder_socket = socket(AF_INET, SOCK_STREAM)
runs on specified IP Adress and Port Number
creates UDP Socket to register with tracker udp_socket =socket(AF_INET, SOCK_DGRAM)
sends REGISTER message to tracker (filename, ip, portnumber)
recieves message from seeder
stores and registers the seeder
sends AVAILABLE message to tracker (IP, port number)
updates timestamp for seeder status

Protocol Messages:

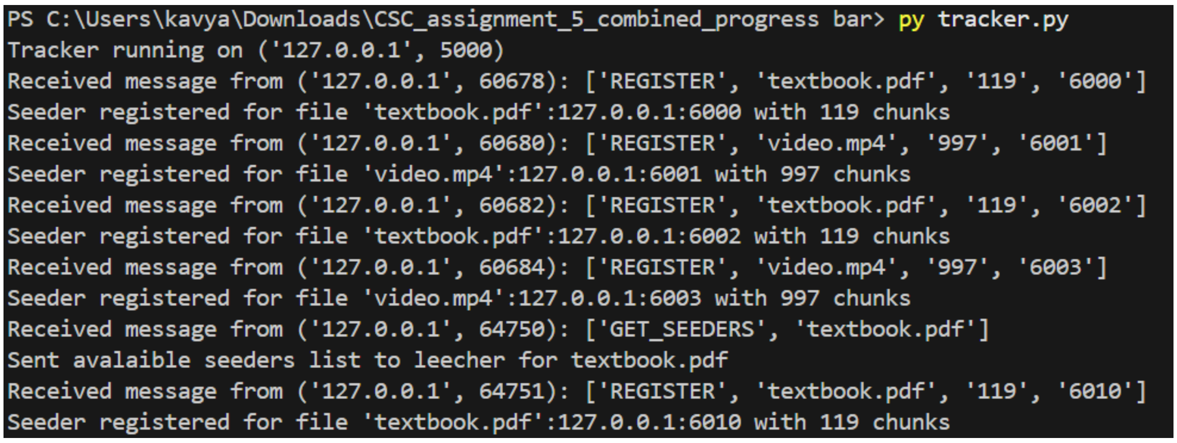
Control type messages:

- Control messages are used for Error Handling. They are found in the tracker, leecher and seeder file in multiple functions. They are used for managing the communication and requesting of data.
- Most messages do not include a header, but they include a body with information/ data that is displayed.

d) Screenshots of the Features

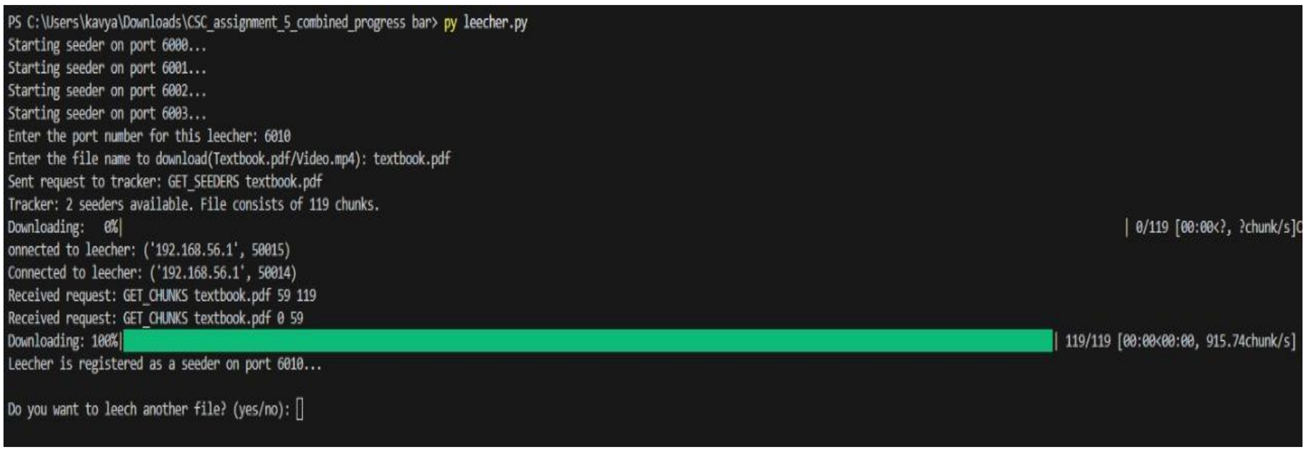
Figure 1 and 2, show how the tracker.py and leecher.py is rrun

- Th tracker.py is run in a separate terminal
- The leecher.py is then run in a separate terminal
- The leecher.py starts off 4 seeder process in the background on ports 6000, 6001, 6002 and 6003
- The program prompts the user to enter a port number to run the leecher and the file name to download. The leecher starts a process on port 6010 and requests "textbook.pdf"
- The requests for seeders is made to the tracker and it obtained that there are **2** seeders available.
- The file consists of 119 chunks which are split between 59 chunks from seeder 1 and 60 chunks from seeder 2.
- A progress bar is instantiated at 0% initially which is then updated to the file size downloaded based off the number of chunks that are downloaded of the total chunks in the file.
- A message is displayed to show if file download was successful.
- Re-seeding takes place and the leecher registers and runs itself as a seeder on the provided port number with the file it just downloaded("textbook.pdf") as can be seen in Figure 1.



```
PS C:\Users\kavya\Downloads\CSC_assignment_5_combined_progress bar> py tracker.py
Tracker running on ('127.0.0.1', 5000)
Received message from ('127.0.0.1', 60678): ['REGISTER', 'textbook.pdf', '119', '6000']
Seeder registered for file 'textbook.pdf':127.0.0.1:6000 with 119 chunks
Received message from ('127.0.0.1', 60680): ['REGISTER', 'video.mp4', '997', '6001']
Seeder registered for file 'video.mp4':127.0.0.1:6001 with 997 chunks
Received message from ('127.0.0.1', 60682): ['REGISTER', 'textbook.pdf', '119', '6002']
Seeder registered for file 'textbook.pdf':127.0.0.1:6002 with 119 chunks
Received message from ('127.0.0.1', 60684): ['REGISTER', 'video.mp4', '997', '6003']
Seeder registered for file 'video.mp4':127.0.0.1:6003 with 997 chunks
Received message from ('127.0.0.1', 64750): ['GET_SEEDERS', 'textbook.pdf']
Sent avalaible seeders list to leecher for textbook.pdf
Received message from ('127.0.0.1', 64751): ['REGISTER', 'textbook.pdf', '119', '6010']
Seeder registered for file 'textbook.pdf':127.0.0.1:6010 with 119 chunks
```

Figure 1: Terminal output of tracker.py

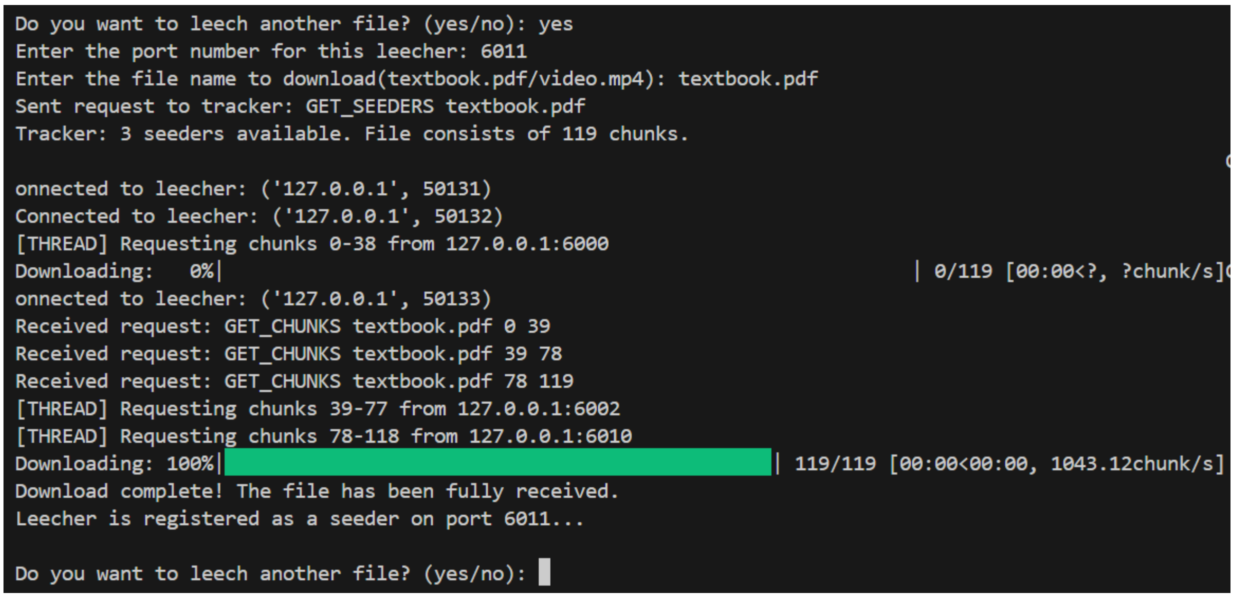


```
PS C:\Users\kavya\Downloads\CSC_assignment_5_combined_progress bar> py leecher.py
Starting seeder on port 6000...
Starting seeder on port 6001...
Starting seeder on port 6002...
Starting seeder on port 6003...
Enter the port number for this leecher: 6010
Enter the file name to download(Textbook.pdf/Video.mp4): textbook.pdf
Sent request to tracker: GET_SEEDERS textbook.pdf
Tracker: 2 seeders available. File consists of 119 chunks.
Downloading:   0%|                                                              | 0/119 [00:00<?, ?chunk/s]C
onnected to leecher: ('192.168.56.1', 50015)
Connected to leecher: ('192.168.56.1', 50014)
Received request: GET_CHUNKS textbook.pdf 59 119
Received request: GET_CHUNKS textbook.pdf 0 59
Downloading: 100%|██████████████████████████████████████████████████| 119/119 [00:00<00:00, 915.74chunk/s]
Leecher is registered as a seeder on port 6010...

Do you want to leech another file? (yes/no): 
```

Figure 2: Terminal output of leecher.py

- Figure 3 shows the following: If the user selects "yes". Another leecher process is run. As seen below a leecher is run on port 6011 and "textbook.pdf" is requested again. Now it can be seen that **3** seeders are available indicating that the reseeding process of the leecher that was ran on port 6010 was successful. File chunks are downloaded from the 3 seeders and the current leecher converts to a seeder.



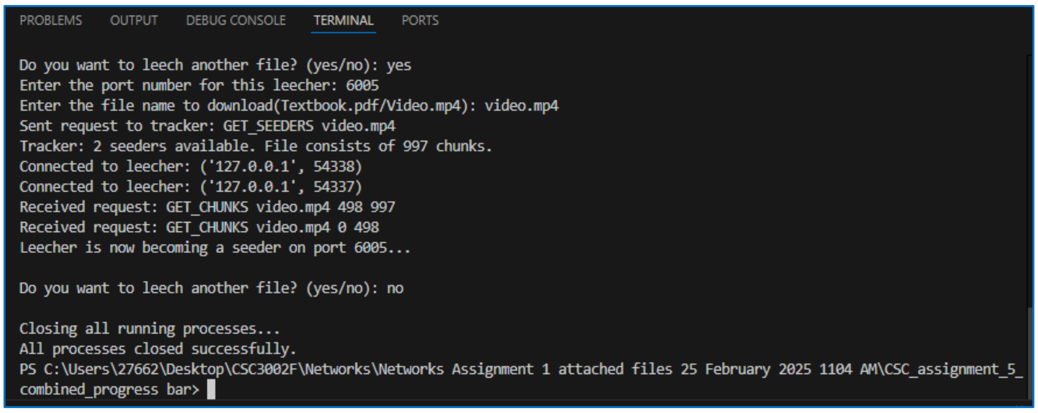
```
Do you want to leech another file? (yes/no): yes
Enter the port number for this leecher: 6011
Enter the file name to download(textbook.pdf/video.mp4): textbook.pdf
Sent request to tracker: GET_SEEDERS textbook.pdf
Tracker: 3 seeders available. File consists of 119 chunks.
                                                                                          C
onnected to leecher: ('127.0.0.1', 50131)
Connected to leecher: ('127.0.0.1', 50132)
[THREAD] Requesting chunks 0-38 from 127.0.0.1:6000
Downloading:   0%|                                            | 0/119 [00:00<?, ?chunk/s]C
onnected to leecher: ('127.0.0.1', 50133)
Received request: GET_CHUNKS textbook.pdf 0 39
Received request: GET_CHUNKS textbook.pdf 39 78
Received request: GET_CHUNKS textbook.pdf 78 119
[THREAD] Requesting chunks 39-77 from 127.0.0.1:6002
[THREAD] Requesting chunks 78-118 from 127.0.0.1:6010
Downloading: 100%|██████████████████████████████| 119/119 [00:00<00:00, 1043.12chunk/s]
Download complete! The file has been fully received.
Leecher is registered as a seeder on port 6011...

Do you want to leech another file? (yes/no): █
```

Figure 3: Terminal output of leecher.py

Figure 4 and 5 show when the leeching process is ended from the user side.

- When a specific seeder is away/deleted it no longer sends its AVAILABLE status, resulting in it being removed by tracker when it times out. You can test this by exiting the leecher.py terminal which removes the seeders and then in the terminal for tracker.py in about 10 seconds the inactive seeders will be removed. **Keep in mind to say no/exit to terminate safely just as good practice.** There is sufficient error handling implemented.

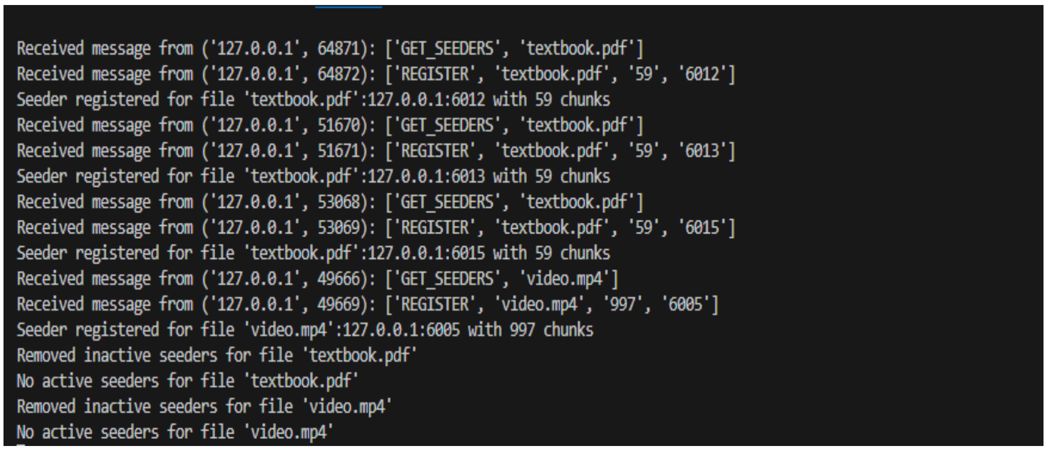


```
Do you want to leech another file? (yes/no): yes
Enter the port number for this leecher: 6005
Enter the file name to download(Textbook.pdf/Video.mp4): video.mp4
Sent request to tracker: GET_SEEDERS video.mp4
Tracker: 2 seeders available. File consists of 997 chunks.
Connected to leecher: ('127.0.0.1', 54338)
Connected to leecher: ('127.0.0.1', 54337)
Received request: GET_CHUNKS video.mp4 498 997
Received request: GET_CHUNKS video.mp4 0 498
Leecher is now becoming a seeder on port 6005...

Do you want to leech another file? (yes/no): no

Closing all running processes...
All processes closed successfully.
PS C:\Users\27662\Desktop\CSC3002F\Networks\Networks Assignment 1 attached files 25 February 2025 1104 AM\CSC_assignment_5_
combined_progress bar>
```

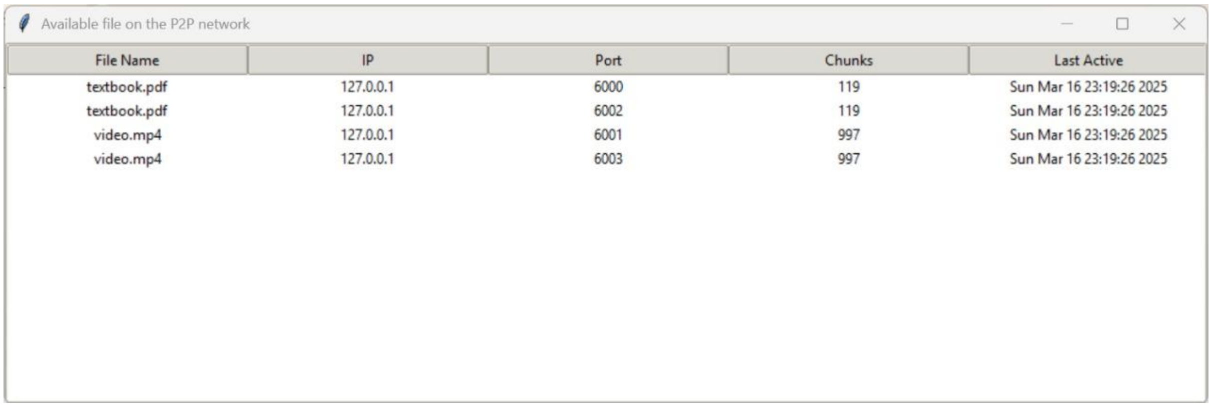
Figure 4: Terminal output of leecher.py



```
Received message from ('127.0.0.1', 64871): ['GET_SEEDERS', 'textbook.pdf']
Received message from ('127.0.0.1', 64872): ['REGISTER', 'textbook.pdf', '59', '6012']
Seeder registered for file 'textbook.pdf':127.0.0.1:6012 with 59 chunks
Received message from ('127.0.0.1', 51670): ['GET_SEEDERS', 'textbook.pdf']
Received message from ('127.0.0.1', 51671): ['REGISTER', 'textbook.pdf', '59', '6013']
Seeder registered for file 'textbook.pdf':127.0.0.1:6013 with 59 chunks
Received message from ('127.0.0.1', 53068): ['GET_SEEDERS', 'textbook.pdf']
Received message from ('127.0.0.1', 53069): ['REGISTER', 'textbook.pdf', '59', '6015']
Seeder registered for file 'textbook.pdf':127.0.0.1:6015 with 59 chunks
Received message from ('127.0.0.1', 49666): ['GET_SEEDERS', 'video.mp4']
Received message from ('127.0.0.1', 49669): ['REGISTER', 'video.mp4', '997', '6005']
Seeder registered for file 'video.mp4':127.0.0.1:6005 with 997 chunks
Removed inactive seeders for file 'textbook.pdf'
No active seeders for file 'textbook.pdf'
Removed inactive seeders for file 'video.mp4'
No active seeders for file 'video.mp4'
```

Figure 5: Terminal output of tracker.py

In Figure 6, a simple GUI that displays the available file names and seeders on the network is shown. The GUI updates every 5 seconds.



File Name	IP	Port	Chunks	Last Active
textbook.pdf	127.0.0.1	6000	119	Sun Mar 16 23:19:26 2025
textbook.pdf	127.0.0.1	6002	119	Sun Mar 16 23:19:26 2025
video.mp4	127.0.0.1	6001	997	Sun Mar 16 23:19:26 2025
video.mp4	127.0.0.1	6003	997	Sun Mar 16 23:19:26 2025

Figure 6: A simple GUI system in the tracker.py